Architecture of E-Health Flanders platform

March 23, 2010

Part I
Documentation Beyound Views

1 Documentation roadmap

1.1 Description of the parts

1.2 Stakeholders

De stakeholders waarop dit document zich focust, zijn de patiënt, de tester en de regering.

Patiënt: De patiënt wil dat wanneer hij bij zijn huisdokter op consultatie gaat, dat de dokter zijn medisch dossier kan raadplegen en aanpassen. Hiervoor moet de patiënt zijn dokter wel de nodige toelatingen geven. Ook wil de patiënt zelf zijn medisch dossier kunnen raadplegen.

Tester: De tester wil de eHealth applicatie grondig testen en alle mogelijke scenarios uitproberen.

Regering: De regering wil kunnen beslissen hoe de toegang tot de medische dossiers wordt geregeld (hoeveel van hun dossier kunnen patiënten zelf zien, welke dokter kan wat aanpassen etc). Ook wil de regering een log kunnen raadplegen waarin alle activiteiten op het eHealth platform gelogd worden. Hieronder vallen onder andere inloggen, medische dossiers raadplegen, medische dossiers aanpassen en voorschriften valideren. Een tak van de regering bestaat uit het RIZIV, die informatie over de uitgeschreven en gevalideerde voorschriften wil bijhouden.

1.2.1 How stakeholders might use the package

Patiënt: De patiënt gebruikt een client-applicatie om het eHealth platform te raadplegen. Wanneer hij op consultatie is, gebeuren de gewenste acties via de client-applicatie van de dokter. Voor de patiënt is de client-kant van deze applicatie dus belangrijk.

Tester: Voor de tester zijn er uitgewerkte sequentiediagrammen aanwezig, die een overzicht geven van hoe de verschillende acties tot stand komen. Daarnaast is in zowel het Client-server diagram als het deployment diagram te zien hoe de verschillende functionaliteiten in afzonderlijke componenten zitten, waardoor het testen van de functionaliteiten afzonderlijk eenvoudig is. Ook het layered view geeft een overzicht van hoe de verschillende componenten samen werken.

Regering: Er is een afzonderlijke component voorzien die de functionaliteiten die de regering vereist, uitvoert. Zo worden alle acties gelogd naar een log, buiten het systeem, en ook de boodschappen naar het RIZIV worden via deze component gestuurd.

2 View template

3 System overview

4 Mapping between views

5 Directory

6 Glossary and acronym list

7 Background, design constraints, and rationale

Part II
Software Architecture Views

1 Module Uses View

1.1 Primary presentation

1.2 Element catalog

1.2.1 Elements and their properties

1.2.2 Relations and their properties

1.2.3 Element interfaces

1.2.4 Element behavior

1.3 Context diagram

1.4 Variability guide

1.5 Architecture background

1.5.1 Rationale

1.5.2 Analysis results

1.5.3 Assumptions

1.6 Other information

1.7 Related view packets

2 C&C Client and Server View: Overview

2.1 Primary presentation

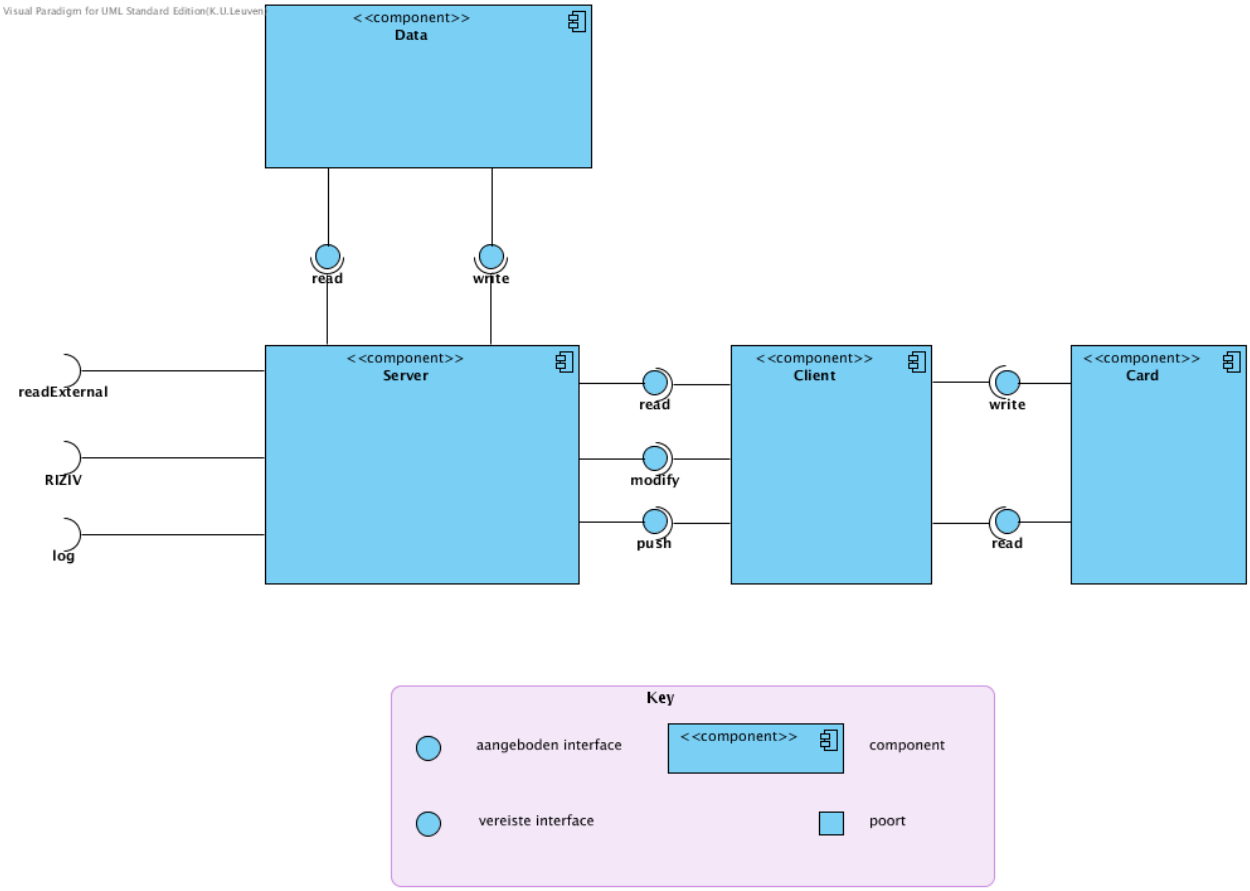

Figure 1: Client and Server View: Overview

2.2 Element catalog

Server De server is de component waar clients verbinding mee maken. De interacties tussen clients en servers gebeuren op een veilige manier. De server is verbonden met clients, de overheid, eventuele externe componenten en de data server. Meer informatie kan gevonden worden in 3 Client and Server View: Server.

Client De client component is de component die gebruikers (dokters, patiënten of apothekers) gebruiken om op een veilige manier met de server te interageren. Meer informatie kan gevonden worden in 4 Client and Server View: Client.

Data De data component is een database waar alle informatie zoals patienten dossiers en dokter data op bewaard zijn. Hoe de data precies wordt opgeslaan is terug te vinden in het 9 Deployment view data.

Card De card component stelt de e-Card van de gebruiker voor. De e-Card bevat gebruikersinformatie alsook twee keys. Een key voor identificatie van de gebruiker en een key voor authenticatie. Naast de gebruikersinformatie en de keys heeft de e-Card ook ruimte voor een aantal voorschriften op te slaan. Meer informatie is terug te vinden in: TODO alsook in enkele interactie diagramma's (zie: 10.4 en 10.5).

2.3 Context diagram

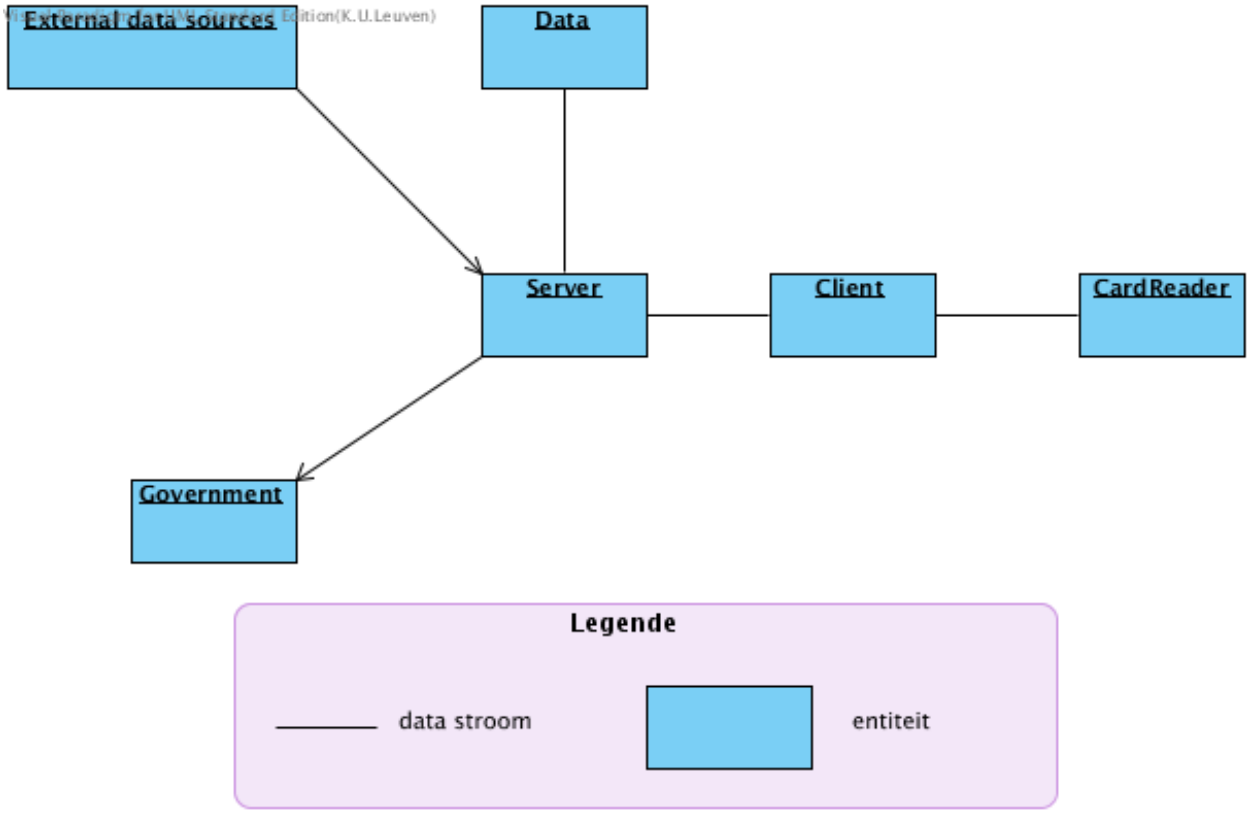

Figure 2: Client and Server View: Context diagram

De belangrijkste extra entiteiten in het context diagram zijn de government en de external data sources. De government component bevat toegang tot de logging en RIZIV database. De external component bevat de validated data sources die gebruikers van de client kunnen opvragen via de server.

2.4 Variability guide

[None]

2.5 Architecture background

2.5.1 Rationale

Enkele design beslissingen die hier te zien zijn is het gebruik van de card reader voor het identificiëren van de gebruikers op de centrale database, meer hierover is te vinden in het ...TODO.

2.6 Related view packets

De volgende client-server views bekijken dit overview diagram in meer detail. Ook in het deployment diagram is extra informatie te vinden vooral dan in verband met de opslag van de global medical records en dokter data.

3 C&C Client and Server View: Server

3.1 Primary presentation

3.2 Element catalog

GMR De GMR is de centrale component in de server component. De GMR delegeert alle inkomende *read*, *modify* of *push* acties naar de correcte andere componenten en doet dit uiteraard in de correcte volgorde. Om alle interacties beter te begrijpen kunnen de Sequence diagrams vaak ook nuttig zijn (zie: 10).

3.2.1 Element interfaces

read Resources provided:
read(data_key, id_key)

De id_key wordt geauthenticeerd en daarna wordt een sessie geopent. De session_key wordt samen met data die bij de data_key hoort teruggegeven.
read(data_key, session_key)

De data met data_key wordt gelezen en teruggeven aan het systeem.

Error handling:
Wanneer de data_key niet geldig is wordt een error teruggegeven.
Wanneer de id_key of session_key geen toegang heeft tot data_key wordt een error teruggegeven.
Wanneer de id_key of session_key niet geldig is wordt een error teruggegeven.

Element requirements:
data_key: de key om de data op te vragen.
id_key: de key die gebruikt wordt om de uitvoerder van de actie op te vragen.
session_key: de key die gebruikt wordt om een sessie te identificiëren.

Rationale and design issues:
Het openen van een sessie wanneer de eerste actie wordt uitgevoerd met een id_key kan ook aangepast worden om een expliciete login te doen. Dan zou de client eerst moeten inloggen met de id_key en dan kan hij later acties uitvoeren met de session_key.
Voor bepaalde acties kan men verwachten dat altijd een id_key wordt meegegeven.
Zie ook 9 Deployment view data voor het ophalen van data. Textuele data wordt altijd eerst teruggegeven, daarna pas andere data (media bestanden zoals figuren, geluid, video).

modify Resources provided:
modify(data_key, data, id_key)
modify(data_key, data, session_key)
Het inloggen en valideren gebeurt zoals beschreven bij **read**. De aanpassing van de data verloopt door het mergen van data met de bestaande data. Wanneer de GMR een antwoord krijgt van de database dat alles correct is geschreven verstuurt de GMR een antwoord naar de client dat alle data correct is geschreven. Deze handelingen gebeuren asynchroon.

Error handling:
Gelijkaardige errors zoals bij **read**.
Wanneer het mergen van de data faalt zal een error teruggegeven worden.

Element requirements:
data_key: de key om de data op te vragen.
data: een patch van data die moet aangepast worden op locatie data_key.

id_key: de key die gebruikt wordt om de uitvoerder van de actie op te vragen.
session_key: de key die gebruikt wordt om een sessie te identificiëren.

Rationale and design issues:
Zie ook **read**.
Het schrijven van data gebeurt met behulp van een patch. Dit verkleint de hoeveelheid data die moet verstuurd worden aanzienelijk en heeft dan ook een belangrijk snelheids voordeel.
In het geval dat een merge actie zou falen, kan aan de laatste client gevraagd worden om de merge manueel te doen, of om een volledig document te verzenden.

push Resources provided:
push(data_key, data, id_key)
push(data_key, data, session_key)
Het inloggen en valideren gebeurt zoals beschreven bij **read**. Het pushen van de data gebeurt gelijkaardig zoals bij **modify**.
Error handling:
Gelijkaardige errors zoals bij **read**.

Element requirements:
data_key: de key om de data op te vragen.
data: data die moet toegevoegd worden aan document met key: data_key.
id_key: de key die gebruikt wordt om de uitvoerder van de actie op te vragen.
session_key: de key die gebruikt wordt om een sessie te identificiëren.

Rationale and design issues:
Zie ook **read**.
Hier kan geen patch gestuurd worden omdat een push actie enkel toelaat nieuwe data toe te voegen aan een dossier.
Alhoewel push ongeveer gelijkaardig is aan modify hebben we er toch voor gekozen om deze toe te voegen. Met behulp van push kunnen bepaalde personen toch toegang krijgen tot het schrijven van data bij een dossier zonder dat dossier te kunnen lezen.

grantAccess Resources provided:
grantAccess(to_key, data_key, action, from_key, period)
Zie grantAccess bij **Policy**. Daar wordt de interface uitgewerkt, de interface hier wordt gewoon gedelegeert naar de interface van **Policy**.

3.2.2 Element behavior

De GMR component delegeert alle inkomende acties. Dat doet hij door inkomende berichten door de security component te laten decrypten. Dan krijgt hij onmiddellijk bevestiging dat de data compleet en correct is en van de een geauthenticeerde gebruiker komt. Indien dit niet het geval is stuurt de GMR een error terug naar de client.
Daarna wordt gecontroleerd of er al een sessie bestaat en of die al dan niet geldig is.
Wanneer dat gebeurd is zal het systeem de oproep doorgeven aan de policy component. Deze zal de GMR laten weten of de actie die de GMR binnen krijgt al dan niet toegelaten is.
Als de actie toegelaten is voor de gebruiker zal de GMR de actie uitvoeren en een gepaste respons sturen naar de client.
Als het nodig is worden acties gelogd of verzonden naar het RIZIV.

SessionManager De SessionManager voorziet in session management. Deze component zorgt ervoor dat een gebruiker niet telkens opnieuw hoeft in te loggen wanneer hij een actie naar het systeem uitvoert.

3.2.3 Element interfaces

open Resources provided:
open(id_key)
Opent een nieuwe sessie voor id_key, de session_key wordt teruggegeven.
Element requirements:
id_key: de key waarmee de client zich identificiëert.
Rationale and design issues:
Voorziet in het openen van sessies, waardoor de client niet telkens zijn identificatie key moet opnieuw ingeven.

valid Resources provided:
valid(session_key)
Dit controleert of de session_key nog geldig is, een sessie verloopt na een bepaalde tijd. Er wordt teruggeven als de session_key nog al dan niet geldig is. Als ze nog geldig is wordt de id_key teruggegeven die verbonden is met deze session_key.
Element requirements:
session_key: de key die gebruikt wordt om een sessie te identificiëren.

Rationale and design issues:
Hoe lang een sessie geldig blijft zou kunnen aangepast worden afhankelijk van wie er in het systeem inlogt of afhankelijk van welke soort client. Wanneer de client toepassing op een smartphone draait zou er kunnen gekozen worden om de session_keys sneller te laten vervallen aangezien deze omgeving als minder veilig kan beschouwt worden. Indien we dit willen afdwingen moet wat extra informatie worden meegegeven bij het openen van de sessie.

3.2.4 Element behavior

De SessionManager is verantwoordelijk voor het openen van sessies (*open*) en het controleren of een bepaalde sessie geldig is (*valid*).

Policy Deze component controleert de toegangsrechten van de gebruikers en laat weten welke gebruikers toegang hebben tot welke delen van de data en wat die toegang inhoudt (lezen en of schrijven).

3.2.5 Element interfaces

autorisation Resources provided:
authorize(data_key, action, id_key)
De authorisatie component kijkt of de persoon met identificatie id_key toelating heeft om actie action uit te voeren op document met key data_key. Er wordt teruggegeven of de actie al dan niet is geauthoriseerd.
Element requirements:
data_key: de key verbonden met een data document.
action: een actie die uitgevoerd zal worden, dit kan read, modify of push zijn.
id_key: de key die de uitvoerder van de actie identificiëert.

Rationale and design issues:
zie ook 9 Deployment view data voor toegang tot de dokter database.

grantAccess Resources provided:
grantAccess(to_key, data_key, action, from_key, period)
De persoon verbonden aan from_key geeft toegangrechten voor periode period aan to_key om actie action uit te voeren op het document met key data_key.

Error handling:
De persoon met from_key heeft geen toelating de toegangsrechten voor actie action toe te kennen.
De persoon met to_key kan geen toegangsrechten krijgen om actie action uit te voeren.

Element requirements:
to_key: de identificatie key van de persoon aan wie toegangsrechten worden toegekend. data_key: de key verbonden met een data document.
action: een actie waarvoor rechten worden uitgedeeld. Action kan hier ook zijn het toekennen van toegangsrechten aan een andere persoon. from_key: de identificatie key van de persoon die de toegangsrechten uitdeelt. period: de periode hoe lang de persoon met to_key toegang krijgt tot de data.

Rationale and design issues:
Door de grantAccess toe te voegen kunnen toegangsrechten worden toegekend aan andere personen en kan dit voor een bepaalde tijdsduur. Action is hier niet enkel read, modify of push zodat bepaalde personen ook toelating krijgen om toegangsrechten door te geven. Zo zou een specialist bijvoorbeeld ook toegang kunnen krijgen om read toegang tot het dossier ook door te geven aan een andere specialist.
zie ook 9 Deployment view data voor toegang tot de key database.

3.2.6 Element behavior

De policy component zal authorisatie toekennen aan bepaalde acties, als die persoon de actie mag uitvoeren. Indien dat niet mag zal de authorisatie geweigerd worden. Bepaalde personen hebben ook rechten om nieuwe rechten toe te voegen. Deze worden via *grantAccess* toegevoegd. In de interactie diagramma's is het verloop duidelijk te volgen (zie 10.1).

Security De gedetailleerde uitwerking van de security component is te vinden in 5 Client and Server View: Security.

ExternalActions

3.2.7 Element interfaces

De Element interface component voorziet de interfaces voor het loggen naar de overheid en het verzenden van de RIZIV data naar de overheid.

log Resources provided:
log(data)
De data wordt gelogd naar de overheid toe. De component zorgt dat enkel data die mag gelogd worden verstuurd wordt.

Element requirements:
data: de data om te loggen.

Rationale and design issues:
De data die mag gelogd worden kan eventueel gedefinieerd worden door een externe instantie, in dit geval de overheid. Zie ook 8 Deployment view services voor toegang tot de overheid.

RIZIV Resources provided:
riziv(data)
De data wordt verzonden naar de overheid toe. De component zorgt data alle data die moet verzonden worden, verzonden wordt.
Error handling:
Wanneer de data niet volledig is wordt een error teruggeven.
Element requirements:
data: de data om te verzenden naar het RIZIV.
Rationale and design issues:
Zie ook 8 Deployment view services voor toegang tot de overheid.

3.2.8 Element behaviour

Deze component zal de logging en RIZIV acties uitvoeren naar de overheid toe. In normale omstandigheden zullen deze acties onmiddellijk uitgevoerd worden. In het geval dat de server van de overheid niet kan bereikt worden zullen de acties gequeued worden totdat de server terug beschikbaar is. Op het moment dat de server opnieuw beschikbaar is zal de queue element per element verzonden worden naar de overheid.
Deze queue kan anders uitgewerkt worden voor log en RIZIV.

3.3 Context diagram

Het 2 Client and Server View: Overview diagram kan beschouwd worden als het context diagram voor de server.

3.4 Variability guide

Het inloggen het systeem gebeurt nu impliciet. Op het moment dat de **GMR** een actie binnen krijgt met een id_key zal aan de **SessionManager** gevraagd worden om een nieuwe sessie te openen. Dan zal de sessie_key teruggegeven worden aan de gebruiker. Dit kan ook aangepast worden om expliciet een login te vereisen voordat de eerste actie in het systeem wordt uitgevoerd.
De **ExternalActions** component kan ook zo uitgewerkt worden dat de acties slechts op bepaalde tijdstippen naar de server van de overheid worden uitgevoerd. De acties worden dan altijd gequeued. Op bepaalde tijdstippen worden de queues dan uitgevoerd en leeggemaakt. Er zou ook kunnen rekening gehouden worden met het aantal elementen in de queue. De queue kan dan worden uitgevoerd en leeggemaakt als een bepaalde treshold van elementen bereikt is. Tenslotte is ook een combinatie van beide vorige mogelijk. De queue kan normaal op een bepaald tijdstip leeggemaakt worden tenzij er te veel elementen in de queue zitten; dan zou de queue eerder worden geleegd. Omgekeerd kan het ook dat de queue altijd geleegd worden zodra een bepaald elementen in de queue zitten, maar ook op bepaalde tijdstippen om in het geval van low traffic de berichten niet te lang op de server te houden.
Het is mogelijk verschillende instellingen te gebruiken voor log en RIZIV.
Het is ook mogelijk om de interfaces zo te implementeren dat bepaalde berichten een hogere prioriteit hebben en altijd direct worden verzonden naar de overheid.

3.5 Architecture background

We hebben de **Server** zo gemaakt met een centrale component die alle delegatie doet; de **GMR**. De **GMR** heeft zelf wel geen verantwoordelijkheden buiten het delegeren van alle inkomende acties.

3.6 Other information

[None]

3.7 Related view packets

In 9, het Deployment view data is te zien hoe de connection naar de externe servers tot stand komt.
In de interactie diagramma's (zie 10 zijn ook voorbeelden te vinden van hoe bepaalde interacties precies in hun werk gaan.

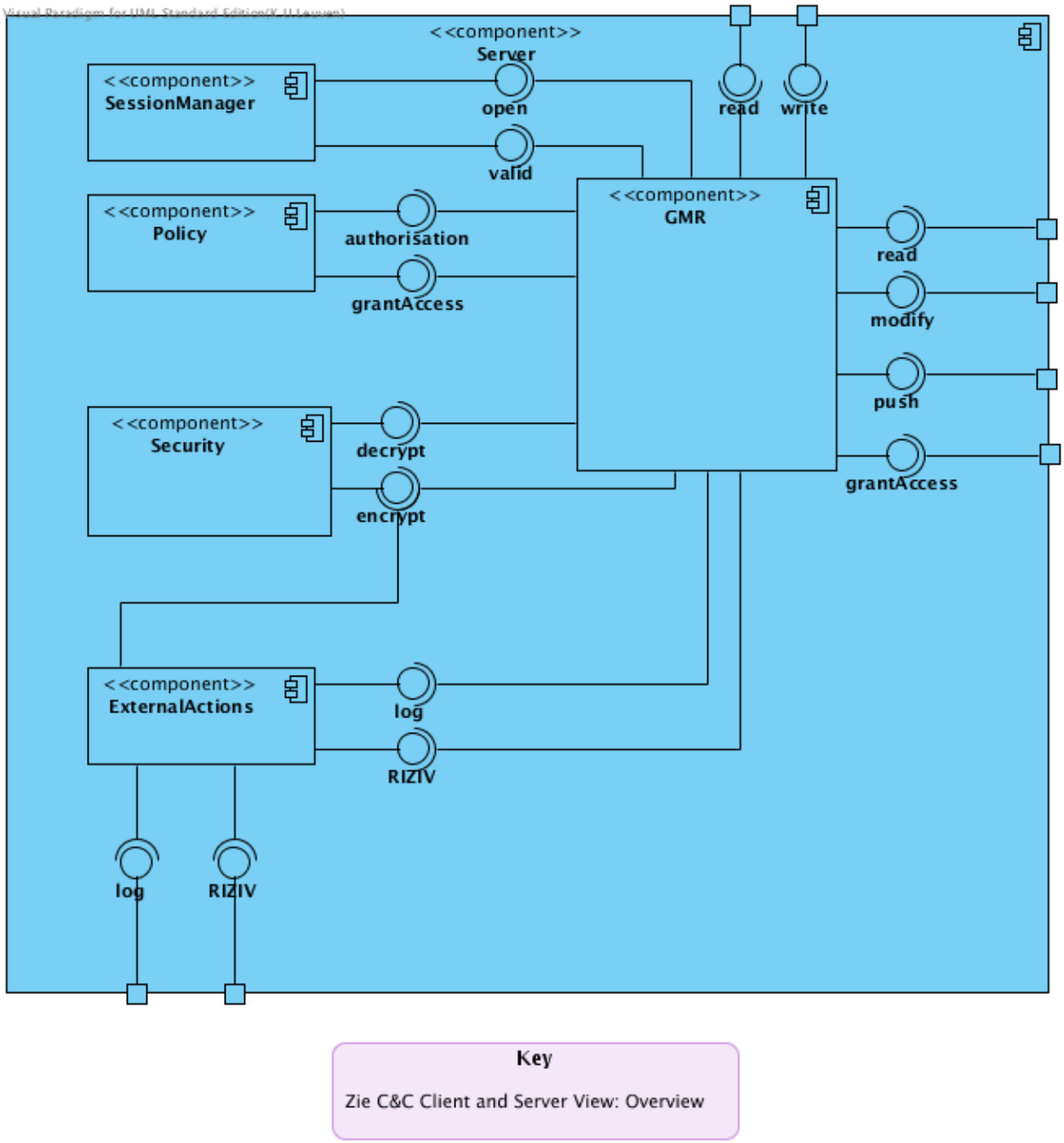

Figure 3: Client and Server View: Server

4 C&C Client and Server View: Client

4.1 Primary presentation

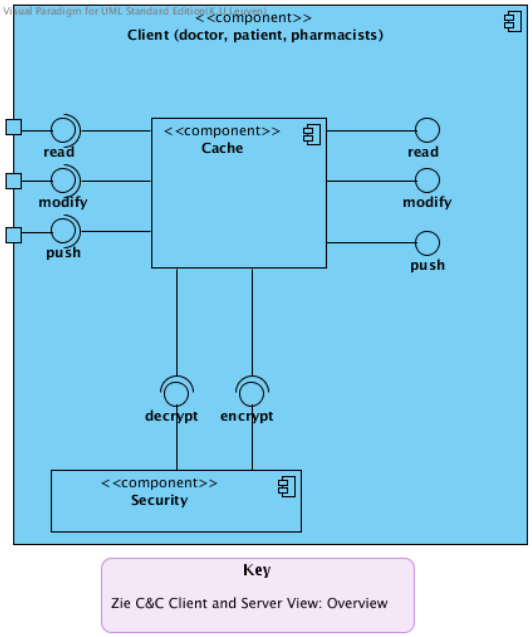

Figure 4: Client and Server View: Client

4.2 Element catalog

Cache De cache component is een lokale cache van bestanden die ook op de server of externe servers aanwezig zijn. Dit element voorziet de toegang tot de data.

4.2.1 Element interfaces

read Resources provided:
read(data_key)
De data verbonden met data_key wordt gelezen (zie ook Elemenet behavior).
Element requirements:
data_key: de key om de data op te vragen.

Rationale and design issues:
Zie hieronder voor een beschrijving van het algemeen gedrag van de cache.

modify Resources provided:
modify(data_key, data)

De aanpassing van de data verloopt door het mergen van data met de bestaande data (zie ook Elemenet behavior).

Error handling:
Wanneer het mergen van de data faalt zal een error teruggegeven worden. Er kan gevraagd worden aan de uitvoerder van de actie om manueel een merge uit te voeren.

Element requirements:
data_key: de key om de data op te vragen.
data: een patch van data die moet aangepast worden op locatie data_key.

Rationale and design issues:
In het geval dat een merge actie zou falen, kan aan de client gevraagd worden om de merge manueel te doen, of om een volledig document te verzenden.

push Resources provided:
push(data_key, data)
Het pushen van de data gebeurt gelijkaardig zoals bij **modify** (zie ook Elemenet behavior).

Element requirements:
data_key: de key om de data op te vragen.
data: data die moet toegevoegd worden aan document met key: data_key.

Rationale and design issues:
Hier kan geen patch gestuurd worden omdat een push actie enkel toelaat nieuwe data toe te voegen aan een dossier.

4.2.2 Element behavior

Wanneer een item uit de cache gelezen wordt (*read*), verstuurt de cache eerst de vraag naar de server als het gegeven item up-to-date is. Als dit het geval is dan wordt het bestand uit de cache aan de gebruiker weergegeven, is dit niet het geval dan stuurt de server het meest recente bestand terug. Dit wordt dan opgeslagen in de cache en dan teruggegeven aan de gebruiker.
Het schrijven van een item (*write*) gebeurt door het item eerst lokaal te schrijven, daarna verstuurt de cache een write operatie naar de server. Deze write operatie verloopt door enkel een patch te sturen van de nieuwe data, het volledige document wordt dus niet weggeschreven maar enkel de wijzigingen. Wanneer de cache een antwoord gekregen heeft van de server dat de schrijfoperatie correct verlopen is mag de cache het geschreven item uit de cache verwijderen.
Een *push* operatie is gelijkaardig aan de schrijf operatie van hierboven. Het enige verschil is dat het in het geval van een *push* gaat over een operatie waarbij de gebruiker geen toegang heeft tot het te schrijven document. Hij kan enkel een data-veld toevoegen, een veld wijzigen of overschrijven is niet mogelijk.
Een laatste belangrijk iets is hoe de cache moet reageren in verband met het falen van het netwerk. Indien de cache geen reactie krijgt van het netwerk mag hij de lokale files teruggeven. In het geval dat deze cache gegevens oud zijn kan dit aan gebruiker gemeld worden. Op het moment dat de cache de server terug kan bereiken zal hij zijn wijzigingen gaan opslaan. Hiervoor wordt een merge algoritme gebruikt.

Security De security component is op zich gelijk aan deze in de server. De gedetailleerde uitwerking is te vinden in 5 Client and Server View: Security.

4.3 Context diagram

Het 2 Client and Server View: Overview diagram kan beschouwd worden als het context diagram voor de client.

4.4 Variability guide

Verschillende implementaties van de Client component zijn mogelijk. Wanneer de client lokaal op een pc wordt uitgewerkt voor een dokter zal de cache waarschijnlijk zo veel mogelijk data bijhouden.
Voor een client van de dokter die bijvoorbeeld op een smartphone draait is het onrealistisch om te verwachten dat deze alle data zal cachen. De client kan dan bijvoorbeeld zo uitgewerkt worden dat de cache van de smartphone iedere morgen voor de huisbezoeken gesynct wordt met de nodige patiënten van die dag.
De centrale pc van de dokter kan een gelijkaardige synchronisatie techniek implementeren voor geplande bezoeken van patiënten.
Nog een ander geval is het geval van de client voor de apotheek of voor de patiënt. In het geval van een client voor de patiënt zal de cache enkel het dossier van de patiënt zelf bijhouden. Voor de apotheek hoeft de cache helemaal geen data te cachen.
Een laatste geval is wanneer de cache op een webclient draait. In dit geval kan de caching aangepast worden naar de mogelijkheden van de webserver. Hier moet dan wel rekening gehouden worden met de wetgeving die centrale opslag van patiënten data verbiedt. Het is mogelijk dat een webserver zoveel zou gaan cachen dat dit zou kunnen gezien worden als een centrale opslag. Omdat dit niet toegelaten is moet men er bij de implementatie van deze cache rekening mee houden. De cache zou zo kunnen aangepast worden om slechts een bepaald aantal documenten bij te houden of enkel de documenten die in een bepaalde tijdspanne opgevraagd zijn.
Het schrijven van de cache naar de server kan ook per client ingesteld worden. Er kan na iedere schrijf actie naar de server geschreven worden, er kan geschreven worden als een dossier wordt gesloten. Het is ook mogelijk dat de server tijdelijk niet beschikbaar is, dan worden alle schrijfacties in een queue gezet en die kunnen dan geschreven worden op het moment dat de server terug beschikbaar is.

4.5 Architecture background

De belangrijkste component van de client, de cache, is gekozen om een aantal redenen. De eerste is het verzorgen van `availability`. Wanneer er een netwerk failure is kan een dokter nog aan alle data die lokaal in de cache zit. In het geval dat de client een dokter is met een lokale pc zal deze cache waarschijnlijk vrij compleet zijn en ook up to date zijn, zeker wanneer het gaat om geplande bezoeken, aangezien de cache voordien al kan synchronizeren met de server.
Een ander belangrijke kwaliteitsvereiste die we hier niet mogen vergeten is de `performance`. Een lokale cache zal een belangrijke snelheids winst opleveren aangezien een stuk minder data over het netwerk moet verzonden worden.
Het wegschrijven van de wijzigingen in plaats van het volledige document bij een *write* operatie naar de server zorgt ook voor een belangrijke snelheids winst. Op deze manier zal veel minder data over het netwerk moeten worden verstuurd.

4.6 Other information

[None]

4.7 Related view packets

Deze client kant moet gezien worden in de volledige overview (zie 2 Client and Server Overview. Daarnaast is ook het Deployment view (**??**, 7, 8, 9) belangrijk.

5 C&C Client and Server View: Security

5.1 Primary presentation

De security component bevind zich zowel in de server als in de client. Deze zorgt voor een veilige communucatie tussen beide.

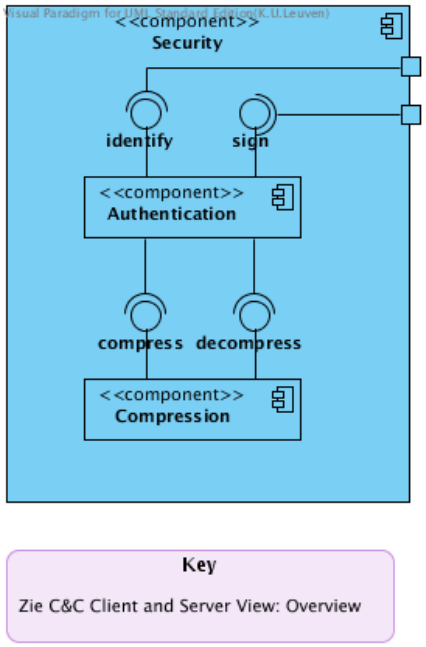

Figure 5: Client and Server View: Security

5.2 Element catalog

Authentication In deze component wordt de identificatie van de personen die zich inloggen op het systeem gecontroleerd. Wanneer iemand een aanvraag doet, wordt nagegaan of de id_key van deze persoon overeenkomt met een gebruiker van het systeem. Ook wanneer een boodschap (aanpassen van GMR bv) wordt verstuurd, wordt deze getekend met de key van de zender. Hiervoor dient de sign-interface.

5.2.1 Element interfaces

identify Resources provided:
identify(id_key)
De gebruiker gelinkt met de opgegeven id_key wordt opgezocht en teruggegeven (zie ook Elemenet behavior).
Element requirements:
id_key: de persoonlijke identificatiekey van de gebruiker.

Rationale and design issues:
Wanneer een gebruiker toegang wil tot het systeem, moet hij dit mogen doen. Hiervoor moet zijn identiteit gekend zijn in het systeem. Door zijn persoonlijke identificatiesleutel op te geven, maakt hij duidelijk dat hij is wie hij beweert te zijn en kan het systeem hem identificeren.

sign Resources provided:
sign(id_key)
wanneer de gebruiker een boodschap wil versturen, signeert hij de boodschap via deze methode (zie ook Elemenet behavior).
Element requirements:
id_key: de persoonlijke identificatiekey van de gebruiker.

Rationale and design issues:
Wanneer de gebruiker een boodschap wil sturen, moet de zender ervan kunnen achterhaald worden. Door de boodschap te signeren met de sign-methode, is de zender ervan bekend bij het systeem en kan achterhaald worden of deze de juiste rechten heeft om deze boodschap uit te voeren.

Compression Boodschappen die tussen de Client en de Server worden verstuurd, worden gecomprimeerd om tijd te besparen en om een extra vorm van beveiliging toe te voegen.

5.2.2 Element interfaces

compress Resources provided:
compress()
De boodschap die wordt verstuurd, wordt gecompresseerd zodat deze onleesbaar wordt voor iemand die deze niet kan decompresseren en zodat de grootte van de boodschap wordt verkleind (zie ook Elemenet behavior).
Element requirements:
/

Rationale and design issues:
Door de boodschappen die worden verstuurd te compresseren, wordt de boodschap zowel beveiligd, als verkleind, waardoor ook de verzendtijd wordt verkleind.

decompress Resources provided:
decompress()
Een boodschap die werd verstuurd en gecompresseerd, moet eveneens worden gedecompresseerd om leesbaar te zijn (zie ook Elemenet behavior).
Element requirements: [None]
Rationale and design issues:
Zie ook compress.

5.2.3 Element behavior

Deze component zorgt dat de communicatie tussen het platform en de clients veilig verloopt. Enerzijds door de boodschap te signeren met een gebruiker zijn identificatiesleutel, waar het enkel geldige sleutels wordt

toegelaten deze boodschappen te sturen. Anderzijds door de boodschappen die verzonden worden te comprimeren. Dit laatste biedt naast veiligheid ook een manier om de communicatie tussen beide te versnellen.

5.3 Context diagram

De 3 Client and Server View: Server en 4 Client and Server View: Client diagramma's kunnen beschouwd worden als context diagram voor de security component.

5.4 Variability guide

De implementatie van zowel de Authentication component als de compression component zal gebeuren met third-party software. Dit is gemakkelijker dan zelf zo een systeem maken en doordat de componenten zo afgezonderd zijn van het systeem, is het later ook gemakkelijk om een nieuw algoritme te voorzien.

5.5 Architecture background

5.5.1 Rationale

zie Element behavior.

5.6 Other information

[None]

5.7 Related view packets

De security component is zowel in de client (4 Client and Server View: Client) als de server (3 Client and Server View: Server) component te zien.

6 Deployment view DMZ

6.1 Primary presentation

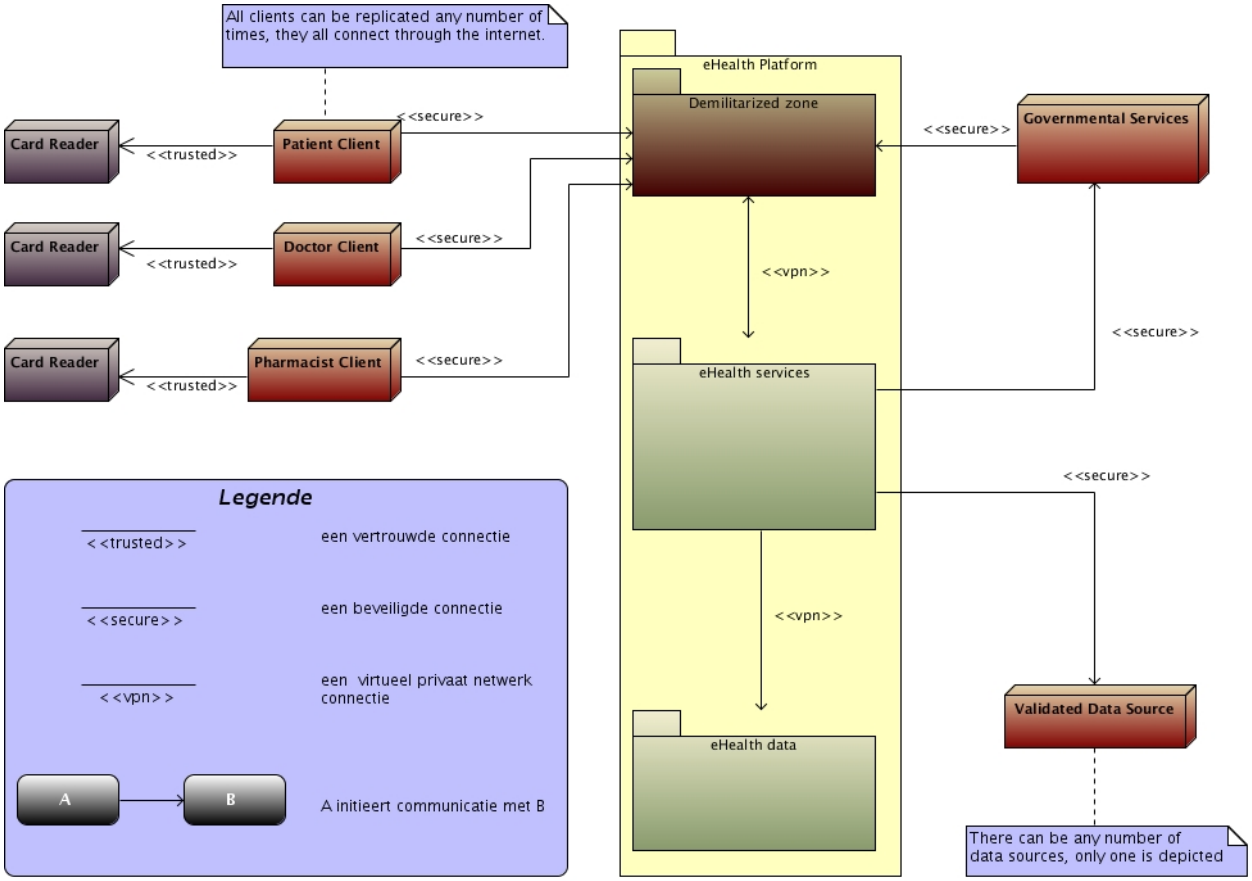

Figure 6: deployment diagram giving a high-level overview of the system

6.2 Element catalog

6.2.1 Elements and their properties

1. **Card Reader**
   Een gespecialiseerde hardware node die in staat is om data van een elektronische kaart te lezen en er ook data op te schrijven. Het kan ook data tekenen en encrypteren met behulp van de keys beschikbaar op de kaart aanwezig in de lezer. Meer info hierover vind je in sectie 7.

2. **Patient Client**
   De PC van een patient. Meer info vind je in sectie 7.

3. **Doctor Client**
   De PC van een dokter. Meer info vind je in sectie 7.

4. **Pharmacist Client**
   De PC van een apotheker. Meer info vind je in sectie 7.

5. **Demilitarized Zone**
   Een gedeelte van het eHealth platform waar publieke services zoals web services en de API zullen zitten. Deze zone vereist geen authenticatie noch authorizatie, het is het toegangspunt tot het platform voor externe gebruikers. Een verdere uitwerking van dit element vind je in sectie 8.

6. **eHealth Services**
   Gedeelte van het eHealth platform dat alle business logica (services) bevat. Een verdere uitwerking van dit element vind je in sectie 8.

7. **eHealth Data**
   Gedeelte van het eHealth platform dat alle data servers bevat. Een verdere uitwerking van dit element vind je in sectie 9.

8. **Validated Data Source**
   Een externe node die gevalideerde medische data aanbiedt. Merk op dat hoewel er slechts 1 node getekend staat, er zo een heel scala aan data bronnen bestaan.

9. **Governmental Services**
   Een externe node waarop de overheid allerhande services aanbiedt zoals het RIZIV en ook een service waarmee belangrijke data gelogd kan worden.

6.2.2 Relations and their properties

Er zijn drie verschillende soorten connecties te onderscheiden :

1. **Vertrouwde connecties**
   Dit zijn de connecties op het diagram aangeduid met het stereotype ¡¡trusted¿¿. Dit zijn connecties die volledig te vertrouwen zijn. Het gaat hier om de connectie tussen een client node en de kaart lezer. Hier moeten we qua security niet al te veel aandacht aan geven aangezien dit gewoon een rechtstreekse connectie is tussen de kaartlezer en de client node, hier kunnen geen aanvallers aan.

2. **Beveiligde connecties**
   Dit zijn de connecties op het diagram aangeduid met het stereotype ¡¡secure¿¿. Dit zijn connecties over het internet die goed beveiligd moeten zijn tegen aanvallers. Dit houdt in dat de twee interagerende nodes moeten weten wie de andere is én dat de communicatie geëncrypteerd is, zodat luistervinken geen nuttige data te pakken krijgen.

3. **Virtueel private netwerk connecties**
   Dit zijn de connectie sop het diagram aangeduid met het stereotype ¡¡vpn¿¿. Deze connecties gaan ook over het internet, maar ze maken deel uit van een virtueel privaat netwerk.

6.2.3 Element interfaces

De interfaces van de verschillende componenten in de nodes staan beschreven in de secties met de client-server views.

6.2.4 Element behavior

Niet van toepassing

6.3 Context diagram

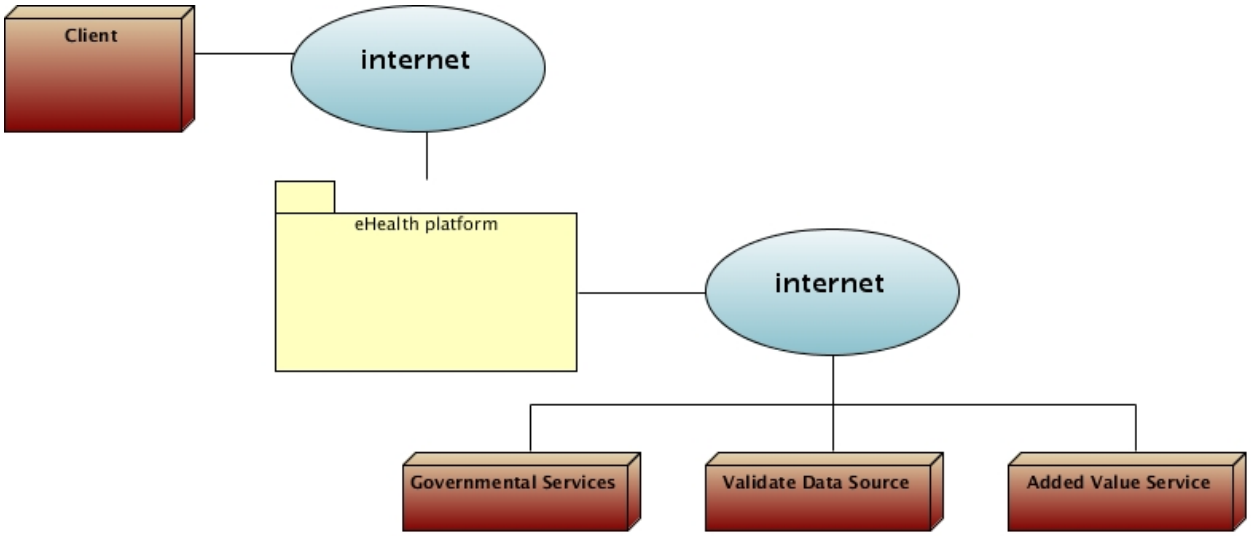

Figure 7: Context diagram for the deployment diagram giving a high-level overview of the system

6.4 Variability guide

6.5 Architecture background

Het eHealth platform wordt opgedeeld in drie delen. Aan de ene kant hebben we een demilitarized zone (DMZ), aan de andere services en data. Door alle clients te laten connecteren langst de demilitarized zone creëren we een centrale plaats vanwaar alle toegang tot het systeem komt. Dit maakt het makkelijker om het systeem te beveiligen en dus ook makkelijker om te voldoen aan de vereiste dat alle toegang tot het platform geauthoriseerd en beveiligd moet worden. Services en data worden ook van elkaar gescheiden wat resulteert in een grotere modulariteit wat het makkelijker maakt om het geheel te implementeren en te testen.

Om ervoor te zorgen dat de demilitarized zone niet omzeild word door een eventuele aanvaller, wordt er gebruikt gemaakt van virtual private networks (VPNs). Dit bevordert de schaalbaarheid, onderhoudbaarheid en variabiliteit van het systeem aangezien het relatief gemakkelijk is om een nieuwe node toe te voegen moest het nodig zijn. Een VPN wordt ook gebruikt voor alle communicatie tussen de services en data lagen. Een nadeel van de DMZ is dat het een bottleneck vormt voor alle toegang tot het platform. Daarom zullen performantie en beschikbaarheid belangrijke drivers zijn voor de verdere decompositie van de DMZ.

Tenslotte zien we dat de verbinding tussen de eHealth services en de overheid slechts in 1 richting gaat, maar vanuit de overheid kan er wel naar de DMZ gegaan worden. Dit komt doordat de overheid via het internet zal connecteren, waardoor we niet kunnen toelaten dat die alle security maatregelen omzeilt.

6.6 Other information

6.7 Related view packets

De verschillende delen in deze view worden nog verder uitgewerkt in aparte secties. Meer info over de verschillende clients is te vinden in sectie 7. Een uitwerking van de DMZ en de services wordt getoond in sectie 8 en over de data in het platform vind je meer in sectie 9.

7 Deployment view clients

7.1 Primary presentation

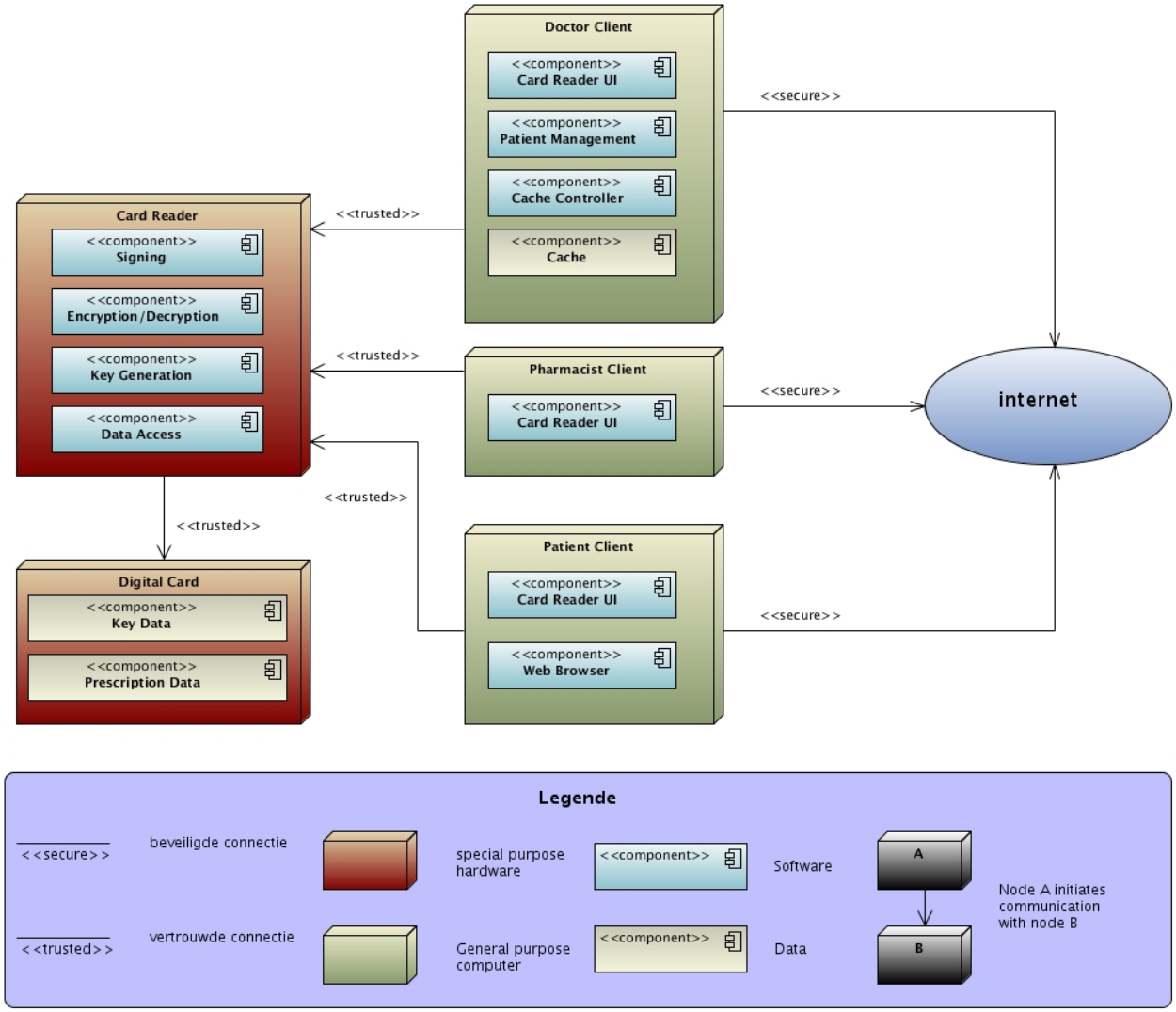

Figure 8: Deployment diagram focused on the various clients in the system

7.2 Element catalog

7.2.1 Elements and their properties

1. **Digital Card**
   Deze node stelt een digitale kaart voor die elke patient en dokter toegewezen krijgt van de overheid.

Deze node zal geen echte software bevatten, maar dient enkel als data opslag. Volgende twee soorten data kunnen we onderscheiden :

(a) **Key Data**
De keys op de kaart. Elke kaart zal een publieke en private sleutel bevatten waarmee getekend en/of geëncrypteerd kan worden met behulp van de elektronische kaartlezer.

(b) **Prescription Data**
De voorschriften die een dokter voorschrijft voor een patient zullen op de kaart bijgehouden worden, zodat deze terug uitgelezen kunnen worden bij het ophalen/bestellen van medicatie bij de apotheker/e-pharmacy.

2. **Card Reader**
De elektronische kaartlezer bevat volgende software componenten :

(a) **Signing**
Software verantwoordelijk voor het tekenen van documenten (voorschriften).

(b) **Encryption/Decryption**
Deze component kan data versleutelen en terug ontcijferen met behulp van de keys die op de kaart staan die in de kaartlezer zit.

(c) **Key Generation**
Sleutels kunnen ook gegenereerd worden. Dit zal ondermeer gebruikt worden voor het tijdelijk toekennen van toegang tot het globaal medisch dossier.

(d) **Data Access**
Deze component is verantwoordelijk voor het ophalen en schrijven van data van/op de elektronische kaart.

3. **Doctor Client**
Stelt een node voor gebruikt door een dokter. Dit kan eender welke soort van node zijn (PC, PDA, laptop, ...). De node bestaat uit volgende componenten :

(a) **Card Reader UI**
De user interface die toegang biedt tot alle functionaliteit van de elektronische kaartlezer.

(b) **Patient Management**
Deze software staat in voor het beheer van het klantenbestand van de dokter.

(c) **Cache Controller**
Component die het ophalen en wegschrijven van patiënt data in zijn/haar globaal medisch dossier controleert. Een meer gedetailleerde uitleg kan gevonden worden in sectie **TODO**.

(d) **Cache**
In de nodes die dokters gebruiken wordt een cache gebruikt met globaal medische dossiers van patiënten van de dokter. Het lezen en schrijven van deze cache wordt beheert door de *Cache Controller*.

4. **Patient Client**
Deze node is de tool van de patient. Het is gewoon een ordinaire computer, laptop of PDA. De enige custom software die deze node nodig heeft is software die de node in staat stelt om te communiceren met de elektronische kaartlezer. Voor de rest kan een patiënt alle nodige activiteiten doen met behulp van een normale web browser, die wel in staat moet zijn om met geëncrypteerde verbindingen om te gaan.

5. **Pharmacist Client**
Node voor een apotheker. Weer is de enige custom software nodig, software om te communiceren met de kaartlezer, alle andere activiteiten kunnen met een gewone web browser uitgevoerd worden.

7.2.2 Relations and their properties

Zie sectie 6.2.2.

7.2.3 Element interfaces

niet van toepassing

7.2.4 Element behavior

niet van toepassing

7.3 Context diagram

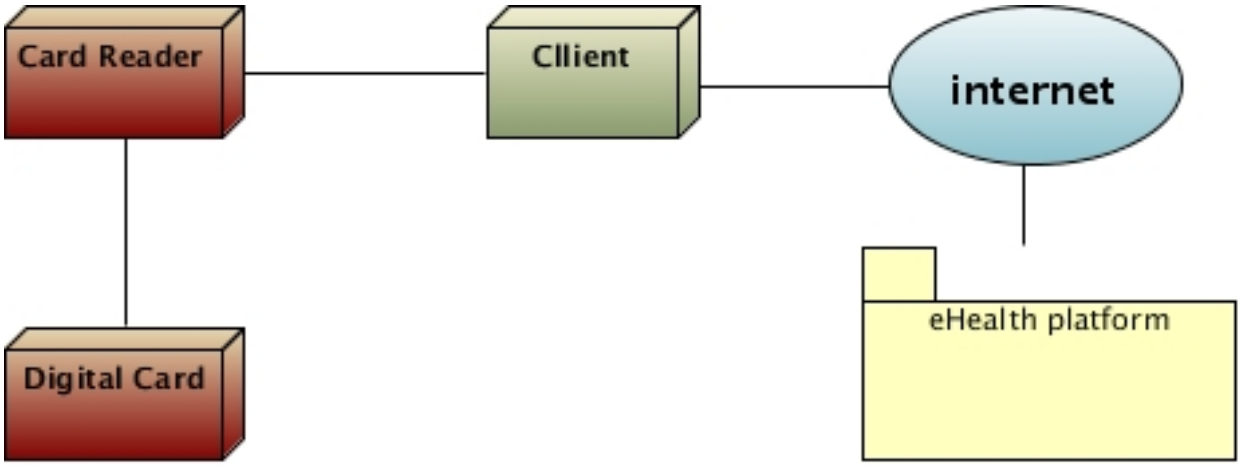

Figure 9: Context diagram for de deployment view focused on the clients

7.4 Variability guide

7.5 Architecture background

Een dokter moet in staat zijn om ten alle tijden de globaal medische dossiers van zijn patiënten in te kijken, ook wanneer hij geen actieve internetverbinding heeft en dus niet met het eHealth platform kan communiceren. Daarom is het belangrijk dat een dokter een lokale cache heeft. Meer informatie over de werking van deze cache vind je in sectie **TODO**. Ook één van de vereisten is dat voorschriften aangemaakt en gevalideerd kunnen worden binnen de 3 seconden , verder moeten voorschriften ook door een dokter aangemaakt kunnen worden, zelfs in geval van een gedeeltelijke systeem of netwerk faling. Daarom zorgen we ervoor dat voorschriften op de elektronische kaart van de patient gezet zullen worden. De dokter kan dan ook wanneer hij offline is het voorschrift creëren (de registratie van het voorschrift in het GMR van de patient wordt dan gequeued tot de dokter terug online is) en de apotheker is in staat om het voorschrift te valideren, ook wanneer die offline is. Doordat dit enkel lokale bewerkingen inhoudt, is de 3 seconden vereiste zeker haalbaar.

7.6 Other information

7.7 Related view packets

Via het internet maken alle clients verbinding met het eHealth platform, dit gedeelte van het systeem vind je gedetailleerder terug in sectie 8.

8 Deployment view services

8.1 Primary presentation

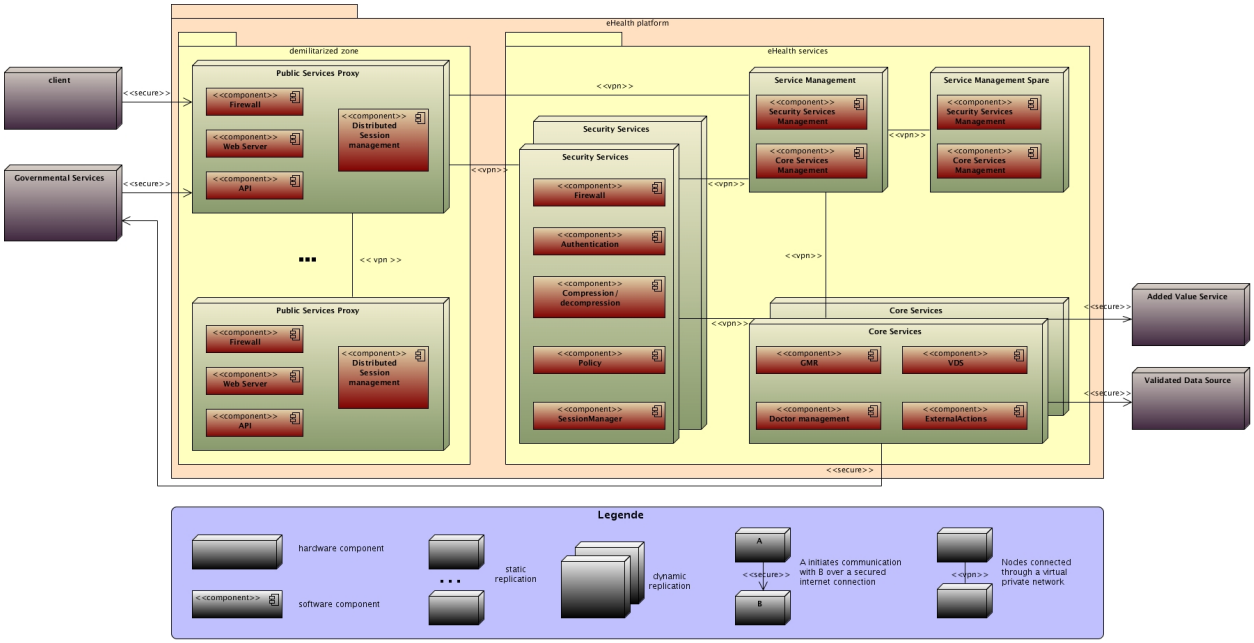

Figure 10: Deployment diagram focused on the services in the platform

8.2 Element catalog

8.2.1 Elements and their properties

1. **Client**
   Stelt de client nodes voor zoals getoond in sectie 7. Hoewel er maar 1 node getoond is, zijn er in de praktijk zeer veel verschillende clients.

2. **Governmental Server**
   Server die de nodige services van de overheid voorziet.

3. **Validated Data Source**
   Stelt de *validate data sources* voor, in de praktijk zijn er meerdere *validated data sources* hoewel er op het diagram slechts 1 getoond wordt.

4. **Added Value Service**
   Stelt de *added value services* voor, weer zullen dit er meerdere zijn.

5. **Public Services Proxy**
   Deze node doet dienst als proxy, zoals z'n naam doet vermoeden. Meer specifiek is het een *reverse proxy*. Deze wordt ingezet om de belasting vanuit het internet naar de security services nodes gelijkmatiger te verdelen, zowel om beveiligings als om *loadbalancing* redenen. Buiten dat, bevat deze node ook

een web server en biedt die de API aan aan externe *added value services*. Deze nodes worden statisch gerepliceerd. De public services proxy bevat volgende software componenten :

(a) **Firewall**
Deze firewall is de eerste verdedigings linie voor het platform en dient het verkeer naar de DMZ toe te filteren zodat enkel connecties specifiek naar de DMZ doorgelaten worden.

(b) **Web Server**
De web server biedt de publieke web services van het systeem aan. Indien dit om zeer algemene data gaat, dient het verkeer niet verder te komen dan de DMZ. Is er echter medische data nodig, dan zal de web server requests doorsturen naar de services laag in het platform.

(c) **API**
Er moet een API voorzien worden voor externe services die gebruik willen maken van het platform. Deze component voorziet deze API en zal alle calls doorsturen naar de services laag van het platform.

(d) **Distributed Session Management**
Het is ongewenst dat een gebruiker zich voor elke actie die hij/zij wil ondernemen terug opnieuw moet inloggen. Er zullen dus sessies bijgehouden moeten worden. Het is niet zo veilig dit te doen binnen de DMZ, dus de authenticatie sessie - data zal bijgehouden worden in de *security services* nodes in de services laag (zie bij security services hier onder). Er zullen meerdere *security services* nodes zijn, dus zal het verkeer van een bepaalde gebruiker gedurende de hele sessie langst dezelfde *security services* node moeten gaan. Het kan zijn, door de structuur en werking van het internet, dat gedurende dezelfde sessie niet dezelfde proxy gebruikt wordt. Om dus toch te kunnen garanderen dat voor één bepaalde sessie steeds dezelfde *security services* node gebruikt wordt, wordt deze component geïntroduceerd. Het enige wat die moet doen is bijhouden welke sessie bij welke *security services* node hoort en dit ook doorgeven aan de andere actieve *Public Services Proxies*.

6. **Security Services**
Deze node vomt de tweede lijn van verdediging voor het platform. Alle verkeer die de core services van het platform wil gebruiken zal langst hier moeten. Het zorgt ervoor dat enkel geauthenticeerde requests met de juiste authorisatie toegelaten worden. Deze node zal dynamisch gerepliceerd worden, later wordt hier wat verder op in gegaan. Volgende software componenten worden op deze node geplaatst :

(a) **Firewall**
Deze firewall zal ervoor zorgen dat enkel connecties vanuit de DMZ toegelaten worden binnen de services laag van het eHealth platform.

(b) **Authentication**
Component die de authenticatie van gebruikers zal afhandelen. Zie sectie **TODO** voor meer informatie.

(c) **Compession/Decomporession**
Component die de compressie en decompressie van data zal afhandelen. Weer biedt sectie **TODO** meer gedetailleerde informatie.

(d) **Policy**
Deze component zal verantwoordelijk zijn voor zowel de authorisatie van requests, als ook voor het beheren van access policies, zoals het geven van toestemming aan dokters om het globaal medisch dossier van een patient in te kijken. Sectie **TODO** legt dit verder uit.

(e) **SessionManager**
We willen niet dat een gebruiker zich voor elke nieuwe request opnieuw moet inloggen. We zullen dus sessie data nodig hebben. Deze component zal het beheer van die sessie data controleren.

7. **Core Services**
   Deze node bundelt alle kern services samen die nodig zijn voor de werking van het eHealth platform. Volgende componenten in deze node stellen al die services voor :

   (a) **GMR**
   Component die alle functionaliteit aanbiedt voor het lezen, schrijven en aanpassen van data in globaal medische dossiers.

   (b) **Doctor Management**
   Beheert alle data gerelateerd met dokters. Registratie van dokters en de registratie van patiënten van dokters gebeurt aan de hand van deze component.

   (c) **VDS**
   Dit stuk software zal alle functionaliteit aanbieden om data te kunnen ophalen van de *validated data sources*, alsook functionaliteit voor het toevoegen van validated data sources aan het systeem. Wanneer zo'n gegevensbron toegevoegd dient te worden, zal die gegevensbron aangeboden moeten worden met een interface die vastgelegd wordt door ons systeem. **TODO : nog meer over die interface zeggen?**

   (d) **ExternalActions**
   Component die de link met de *Governmental Services* voorziet (loggen en RIZIV).

8. **Service Management *(Spare)***
   Deze node zal verantwoordelijk zijn voor het beheren van de verschillende services in het systeem. Zoals je kan zien op het diagram worden de security services en de core services beiden apart dynamisch gerepliceerd. Dit houdt in dat het aantal actieve processen gedediceerd voor deze twee onderdelen afhankelijk zal zijn van de belasting van het systeem. Deze node controleert dit alles. Het bevat twee aparte componenten, 1 voor de security services en 1 voor de core services. De node zal ook verantwoordelijk zijn voor het registreren en het deregistreren van de verschillende security services processen aan de *Public Services Proxies* zodat deze weten welke er beschikbaar zijn.

8.2.2 Relations and their properties

Zie sectie 6.2.2 voor meer info over de verschillende connecties tussen de nodes.

8.2.3 Element interfaces

Niet van toepassing.

8.2.4 Element behavior

Wanneer een request gemaakt wordt naar het platform, zal deze via de *Public Services Proxy* moeten. Deze zal moeten beslissen naar welke van de actieve *Security Services* servers de request doorgestuurd zal worden. Zoals eerder vermeld zal dit voor een bepaalde sessie steeds dezelfde moeten zijn om authenticatie bij te kunnen houden. Dus als de *Public Services Proxy* een request krijgt van een sessie die reeds in het *Distributed Session Management* geregistreerd staat, moet er geen keuze gemaakt worden. Wanneer een request binnenkomt die nog niet geregistreerd is, zal de *Public Services Proxy* een korte ping-boodschap sturen naar de 3 *Security Services* servers die het minst in het *Distributed Session Management* gekoppeld zijn aan een sessie. De eerste die hierop antwoord zal dan gekozen worden om de request naar door te sturen.

De *Public Services Proxy* moet natuurlijk ook weten welke *Security Services* servers er beschikbaar zijn. Hiervoor is de *Service Management* server verantwoordelijk. Deze zal aan de hand van een ping-echo techniek elk van de *Security Services* servers controleren. Wanneer 1 van die servers niet meer antwoord, betekent dit dat die node faalt en zal dit doorgegeven worden aan de *Public Services Proxies*. In de ping-echo boodschap

zullen de *Security Services* servers ook hun huidige load aangeven. Wanneer de *Service Management* server merkt dat alle servers zwaar geladen zijn.zal deze beslissen om een nieuwe op te starten. Dit zal dan ook doorgegeven worden aan de *Public Services Proxies*.

Dit zelfde systeem wordt gebruikt voor de connectie tussen de *Security Services* servers en de *Core Services* servers. Een *Security Services* zal eerst naar 3 verschillende *Core Services* servers een korte ping-boodschap sturen en vervolgens diegene die het snelst antwoord kiezen om verder mee te werken. De *Service Management* server zal er nu ook weer voor zorgen dat de *Security Services* servers weten met welke *Core Services* servers ze kunnen connecteren en deze laatste ook monitoren en dynamisch toevoegen indien nodig.

8.3 Context diagram

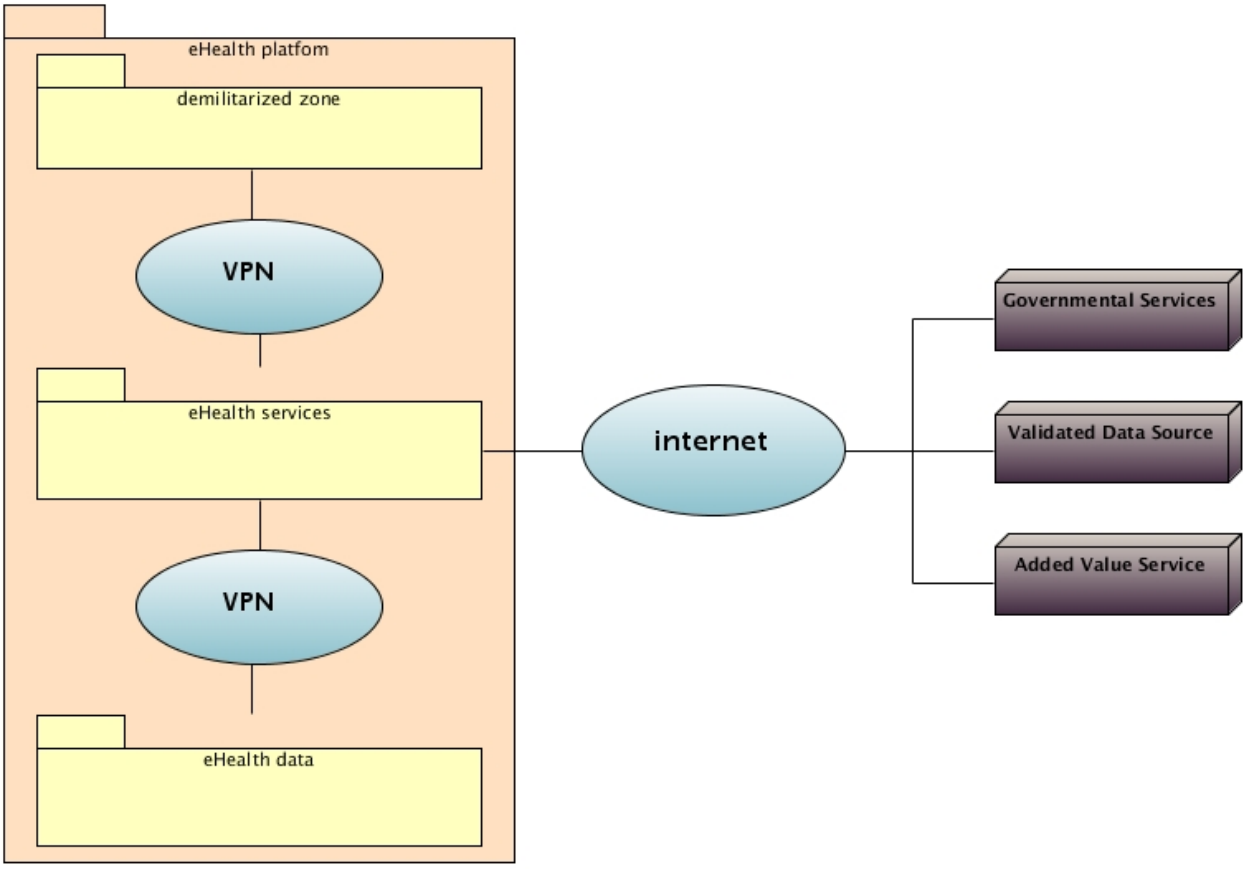

Figure 11: Context diagram for the deployment diagram focused on the services in the platform

8.4 Variability guide

8.5 Architecture background

Omwille van de zware beschikbaarheids vereisten - maximum 2 minuten downtime per jaar - hebben we alle nodes gedupliceerd, zodat er zeker niet één *single point of failure* in het systeem zit. In het gedeelte bescreven

in deze sectie worden 3 verschillende replicatie-technieken gebruikt. De *Public Services Proxy* servers worden statisch gerepliceerd. Dit betekent dat het aantal servers niet door het systeem zelf zal gewijzigd worden. Natuurlijk kunnen er wel manueel nieuwe servers toegevoegd worden.

De replicatie-techniek gebruikt voor voor de *Security Management* servers is een *spare*. De gewone server zal er tijdens het lopen voor zorgen dat de staat van de reserve steeds in real-time mee ge-update wordt. Dit zal gebeuren op een regelmatig (kort) interval. Wanneer de reserve een aantal van die updates mist, neemt deze aan dat de primaire *Security Management* server gefaald heeft en zal die voortaan de functies van de primaire overnemen.

Ten slotte wordt er ook gebruik gemaakt van dynamische replicatie voor zowel de *Security Services* als de *Core Services*. Dit zorgt voor een goede schaalbaarheid van het systeem aangezien het systeem zich zal aanpassen aan de load.

Performantie is natuurlijk ook belangrijk. Bepaalde gegevens moeten binnen bepaalde tijdspannes beschikbaar zijn. Daarom gebruiken we de methode hierboven beschreven om van de *Public Services Proxies* naar de *Security Services* en van de *Security Services* naar de *Core Services* te gaan. We zullen telkens eerst een korte ping-boodschap sturen naar minstens 3 nodes en op basis van de antwoorden op deze ping een keuze maken. Dit doen we om ervoor te zorgen dat we niet te veel tijd verspillen met het wachten op een timeout met een falende component. Ook is dit een primitieve vorm van dynamisch loadbalancing, het antwoord dat het eerst terugkomt zal gewoon de snelste (en dus waarschijnlijk minst belaste) route volgen.

8.6 Other information

Het aantal *Public Services Proxies* moet nog bepaald worden. Hiervoor zal eerst onderzoek gedaan moeten worden naar de verwachte load van het systeem en moeten we weten welke systemen we beschikbaar hebben.

8.7 Related view packets

9 Deployment view data

9.1 Primary presentation

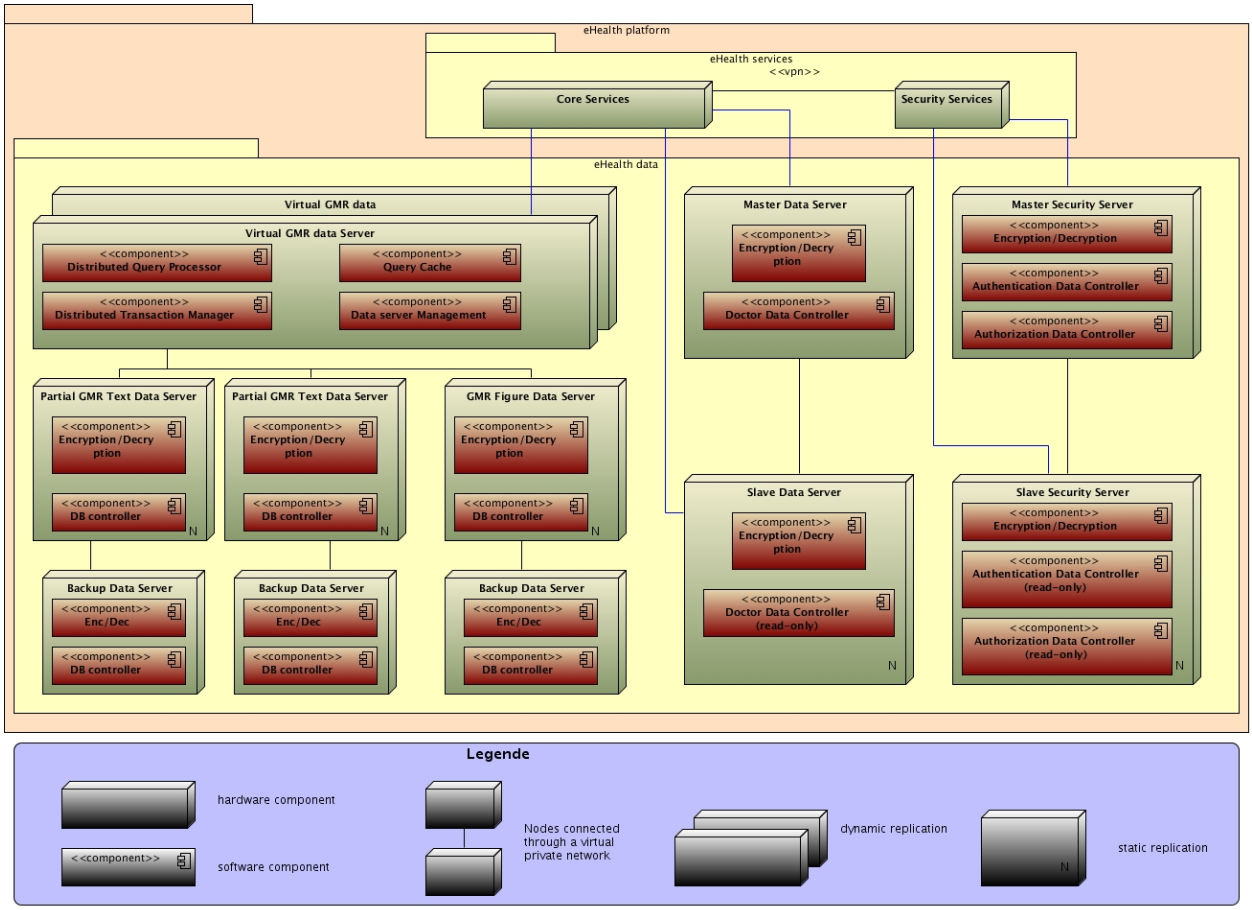

Figure 12: Deployment diagram focused on the data storage in the platform

9.2 Element catalog

9.2.1 Elements and their properties

1. **Core Services**
   Node die alle kern services van het systeem bundelt. Meer info vind je in sectie 8.

2. **Security Service**
   Node die alle security services van het systeem bundelt. Meer info vind je in sectie 8.

3. **Virtual GMR Data Server**
   Deze server zal alle GMR data als een coherent geheel beschikbaar stellen. Aangezien volgens de wetgeving niet alle data van één enkel GMR op 1 plaats opgeslagen mag worden, hebben we deze node

nodig die de nodige delen van een opgevraagd GMR zal ophalen op de aparte data servers. Deze node zal dynamisch gerepliceerd worden volgens de load, net zoals de *Security Services* en de *Core Services* in de services laag. Volgende componenten zitten in deze node :

(a) **Distributed Query Processor**
Deze component zal ervoor zorgen dat queries voor GMR data correct opgesplitst zullen worden in aparte queries voor de aparte data servers. Het laat die aparte queries hun werk doen en zal achteraf alle verkregen dat terug bijeenzetten tot een geheel dat dan als antwoord dient op de orginele query.

(b) **Distributed Transaction Manager**
Dit stuk software is verantwoordelijk voor de consistentie van de GMR data. Wanneer er een query uitgevoerd moet worden die gegevens update of wegschrijft, zal deze component er verantwoordelijk voor zijn dat alle subqueries uitgevoerd worden. Indien er 1 van de subqueries faalt, zullen de andere subqueries ongedaan gemaakt worden door dit element.

(c) **Query Cache**
Dit element zal tijdelijk de laatste queries die uitgevoerd zijn op de GMR data bijhouden, dit zal nodig zijn voor eventuele herstellingsoperaties bij het falen van 1 van de data servers zoals uitgelegd in sectie 9.2.4.

(d) **Data Server Management**
Dit element biedt functionaliteit om de aparte data servers te beheren.

4. **Partial GMR Text Data Server**
Deze server zal tekstuele GMR data bevatten. Het zal niet alle data van 1 GMR bevatten, maar slechts gedeeltes van die date, aangezien dit bij wet voorgeschreven wordt. Deze servers zullen statisch gerepliceerd worden. De node bevat volgende software componenten :

   (a) **Encryption/Decryption**
   Alle data op deze node dient geëncypteerd opgeslagen worden.Deze component zorgt daarvoor en zorgt er ook voor dat wanneer er data opgevraagd wordt, deze data ontcijferd wordt.

   (b) **DB Controller**
   Algemene database controller die de eigenlijke data access verzorgt.

5. **GMR Figure Data Server**
Net hetzelfde als de *Partial GMR Text Data Server*, behalve het soort van data dat erin opgeslagen wordt is anders.

6. **Backup Data Server**
Deze servers zijn backups van de data servers waarmee ze verbonden zijn.

7. **Master Data Server**
Deze node zal data bijhouden die geen deel is van GMR data, namelijk data over dokters en welke patienten bij welke dokter horen. Deze node is statisch gerepliceerd door middel van een master-slave communicatie. Deze node is de master, wat wil zeggen dat op deze node geschreven en gelezen kan worden. De andere nodes zijn slaves, wat wil zeggen dat daar enkel van gelezen kan worden. Merk op dat hoewel er maar 1 slave node getoond wordt, er meerdere mogelijk zijn. Deze nodes hebben, net zoals de *Partial GMR Text Data Servers*, hun eigen private sleutel waarmee alle data op die node geëncrypteerd en gedecrypteerd kan worden.

8. **Slave Data Server**
Zie item 7.

9. **Master Security Server**
   Deze node zal alle security gerelateerde data bijhouden (authenticatie en authorisatie data). Ook deze node wordt gerepliceerd door middel van een master-slave configuratie en wordt beveiligd door een eigen private sleutel om data mee te versleutelen, net zoals bij de *Master Data Server*.

10. **Slave Security Server**
    zie item 9.

9.2.2 Relations and their properties

Zie sectie 6.2.2 voor een uitleg over de types van connecties.

9.2.3 Element interfaces

Niet van toepassing.

9.2.4 Element behavior

Als er vanuit de *Core Services* data opgevraagd moet worden, zal weer hetzelfde mechanisme in werking treden als bij de communicatie tussen de *Public Services Proxies* en de *Security Services* beschreven in sectie 8.5. Er zal eerst een ping bericht gestuurd worden naar minstens 3 virtuele servers en diegene die het snelst antwoord zal gekozen worden om verder mee te werken.

Wanneer een virtuele data server een query krijgt voor GMR data, zal deze die query opsplitsen in subqueries die de verschillende delen van de data zal ophalen van de verschillende data servers die elk partiële data bevatten. Daarna worden die subqueries uitgevoerd op de correcte data server en de resultaten worden samengegoten tot 1 geheel dat als antwoord dient voor de originele query. Elke query zal door de virtuele data server ook gecached worden. Indien de query data update of toevoegt, zal de *Distributed Session Manager* ervoor zorgen dat de volledige operatie uitgevoerd wordt, of indien dit niet lukt, de query helemaal geen invloed gehad heeft op de data.

Alle data servers hebben hun eigen private sleutel. Wanneer data toegevoegd wordt, zal die data met die sleutel geëncrypteerd worden vooraleer die opgeslagen wordt. Wanneer er dan data opgevraagd wordt, zal met diezelfde sleutel de data gedecrypteerd worden. Deze data mag echter niet ongeëncrypteerd verstuurd worden, dus moet elke request voor data een andere sleutel meegeven waarmee de data eerst geëncrypteerd zal worden alvorens die doorgestuurd wordt.

De GMR data servers hebben elk een backup data server. Op geregelde tijdstippen worden die backups gesynchroniseerd met de data op de servers. Dit wordt geregeld door de *DB Controller* component. Indien 1 van deze GMR data servers zou falen, dan zal de backup-server dit merken doordat er geen synchronisaties meer plaatsvinden. Indien dit het geval is, spreekt de backup de virtuele data servers aan en vraagt alle relevante queries op die uitgevoerd zijn sinds de laatste synchronisatie van de backup. De backup update zijn gegevens en zal vanaf dan dienst doen als GMR data server.

9.3 Context diagram

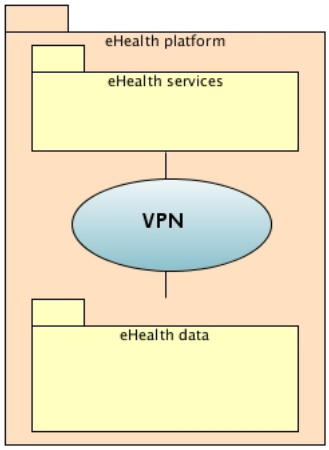

Figure 13: Context diagram for the deployment diagram focused on the data storage in the platform

9.4 Variability guide

Op het diagram, wordt 1 GMR nu opgedeeld in drie stukken : 2 tekstuele stukken en 1 stuk voor andere data (zoals bijv. figuren). Dit kan eventueel nog gewijzigd worden, afhankelijk van de exacte wettelijke voorschriften over de scheiden van de data. Het functioneren van het systeem zal hierdoor niet wijzigen.

9.5 Architecture background

Het is bij wet verplicht om de data voor een bepaald GMR verspreid bij te houden, vandaar dat het mechanisme met de virtuele data servers in het systeem geplaatst wordt. Door een data server management component te voorzien in de virtuele data servers, wordt het mogelijk om het aantal servers uit te breiden indien dit nodig moest zijn, wat de variabiliteit van het systeem bevordert. Het verspreiden van de data is ook zeer gunstig vanuit een security standpunt. Moest het zijn dat een aanvaller de disk op 1 van de servers kan kopiëren, zal deze nog steeds niet zijn met de data die hij ervan af gehaald heeft. Ook wordt de data geëncrypteerd, wat de data nog nuttelozer maakt voor een aanvaller.

Om aan de beschikbaarheids-vereisten te voldoen, wordt er gebruik gemaakt van replicatie. Voor de doktor data en de security data wordt dit gedaan aan de hand van een master-slave configuratie. Hierdoor wordt de load van het systeem verspreid over de master en de vershillende slaves om zo ook aan de performantie-vereisten te kunnen voldoen. Aangezien er slechts 1 master is, zou dit een bottleneck kunnen zijn, maar aangezien de doktor en security data waarschijnlijk niet vaak aangepast zullen worden, zal dit geen problemen geven. Voor de GMR data zou dit wel een performantie bottleneck zijn aangezien deze data veel frequenter aangepast wordt. Daarom voorzien we hier een ander mechanisme. De partiële data servers worden statisch gerepliceerd, zonder hier een master in aan te duiden, er mag naar alle replica's geschreven worden. Wanneer deze servers met elkaar synchroniseren, zal er bij conflicten een merging-techniek gebruikt worden zoals beschreven in sectie **TODO**.

De performantie-vereisten voor het ophalen van tekstuele data en andere data verschillen nogal wat. Om ervoor te zorgen dat het ophalen van tekst nooit moet wachten op het ophalen van figuren, worden tekst en figuren volledig van elkaar gescheiden.

9.6 Other information

9.7 Related view packets

10 Interaction Diagrams

Bij de communicatie tussen de user applicaties en het e-health platform wordt de verzonden informatie steeds ge-encrypteerd en gecompresseerd. Dit wordt in de diagramma's weggelaten om het overzichtelijker te maken.

10.1 Grant access

Wanneer een patient een dokter toegang tot zijn dossier wil geven, voert deze eerst zijn persoonlijke code in om zijn persoonlijke key aan het systeem van de dokter te koppelen. Vanuit de applicatie van de dokter (we veronderstellen hier dat het toegang geven gebeurt bij de eerste consultatie, en niet thuis op voorhand) wordt dan de vraag om de toegangsrechten te wijzigen verstuurd. Ook de persoonlijke toegangscode van de patient wordt gevraagd en nagegaan om te zien of deze de rechten heeft om toegang te verlenen. Wanneer dit in orde is, worden de toegangsrechten tot het dossier zo aangepast dat de dokter er toegang tot heeft voor de periode die patient heeft opgegeven (kan ook permanent zijn).

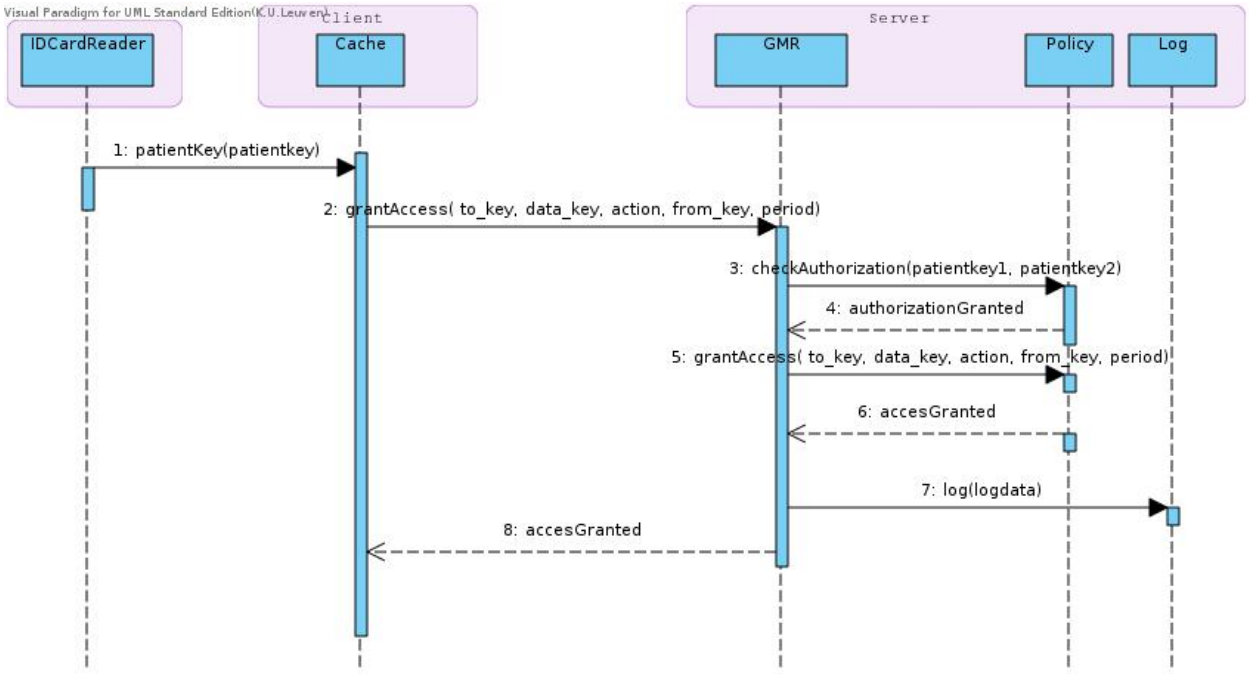

Figure 14: Interaction diagram: Grant acces

10.2 Modify GMR

Wanneer een dokter het dossier van een patient aanpast, doet hij dit eerst op de lokale kopie in zijn cache. De dokter heeft eerst, bij het openen van het dossier, de laatste versie bemachtigd zoals beschreven in het 'access file' diagram. Wanneer hij de wijzigingen hefet opgelagen in zijn cache, worden deze naar de database overgebracht. Er wordt eerst nog gekeken of de dokter ook het recht heeft om te schrijven naar het dossier. Dan worden de wijzigingen opgeslagen in de database. Alle wijzigingen worden op deze manier doorgevoerd, zowel het aanpassen van gegevens, opladen van voorschriften en toevoegen van beeldmateriaal(bv röntgenfotos)

Wanneer de server niet bereikbaar blijkt wanneer er veranderingen aan een file worden gemaakt, worden de veranderingen enkel lokaal bijgehouden. Zodra de server weer bereikbaar is, worden de veranderingen doorgevoerd. In principe kan dit leiden tot een inconsistentie (het dossier werd bijgewerkt door iemand anders terwijl de dokter offline was) maar in de praktijk is dit bijna onmogelijk, aangezien een patiënt meestal slechts 1 huisdokter heeft die zijn dossier lokaal bewaart en wanneer andere dokters toch veranderingen aanbrengen (bv een specialist), dan zal dit aan andere velden van het dossier zijn als die dat de dokter heeft aangepast (bv XRay foto's toegevoegd).

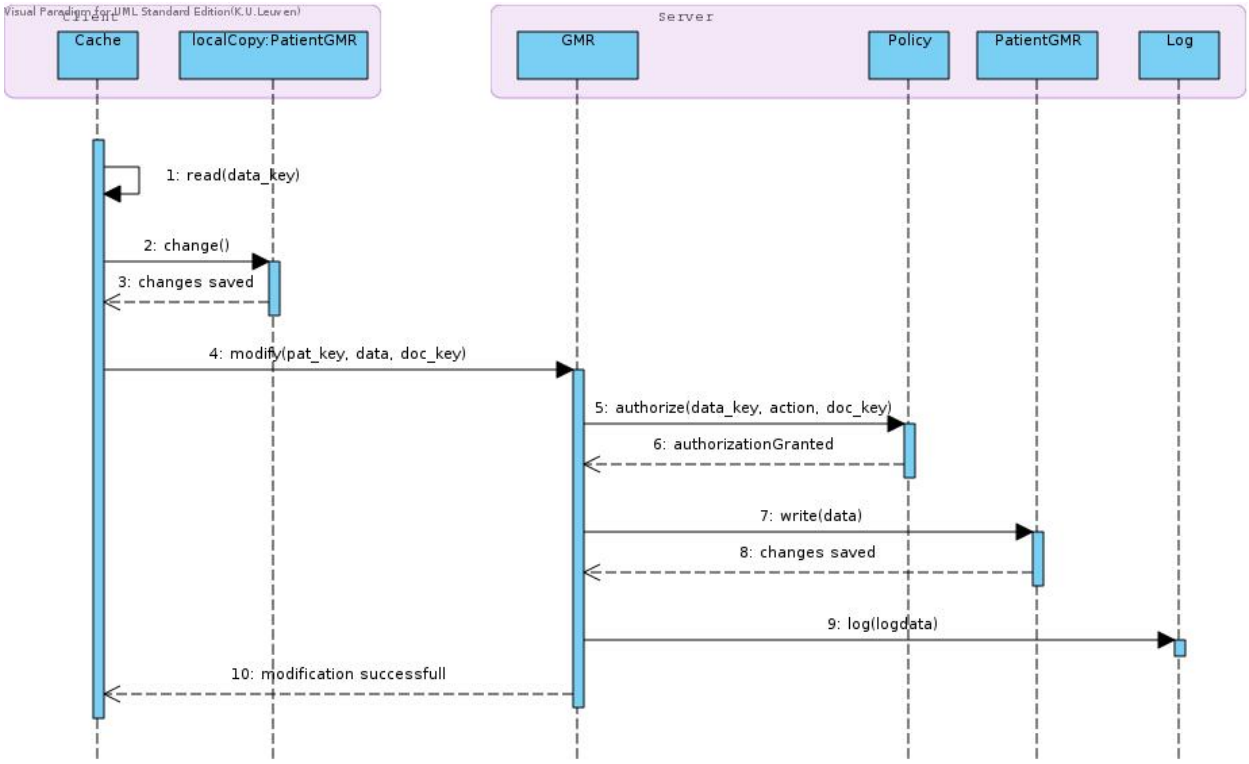

Figure 15: Interaction diagram: Modify GMR

10.3 Retrieve GMR

Wanneer een dokter een medisch dossier van een patient wilt raadplegen, wordt dit dossier eerst in zijn cache opgezocht. Wanneer het hier gevonden wordt, wordt de timestamp van het dossier in de cache vergeleken met de timestamp van het dossier in de database (indien mogelijk). Wanneer het dossier up to date blijkt, wordt gewerkt met de versie uit de cache. wanneer er aanpassingen zijn geweest, wordt het dossier eerst geupdated in de cache. Wanneer het dossier niet aanwezig blijkt in de cache, wordt het dossier opgehaald en opgeslagen in de cache. Deze cache wordt ook gebruikt wanneer de server niet bereikbaar is. Zo kan de dokter toch steeds aan een recente kopie van het dossier van zijn patiënten. Ook de aanpassingen aan de dossiers worden dan enkel lokaal bewaard (zie ook modifyGMR). Zodra de server weer bereikbaar is, worden de veranderingen aan de dossiers ook daar doorgevoerd.

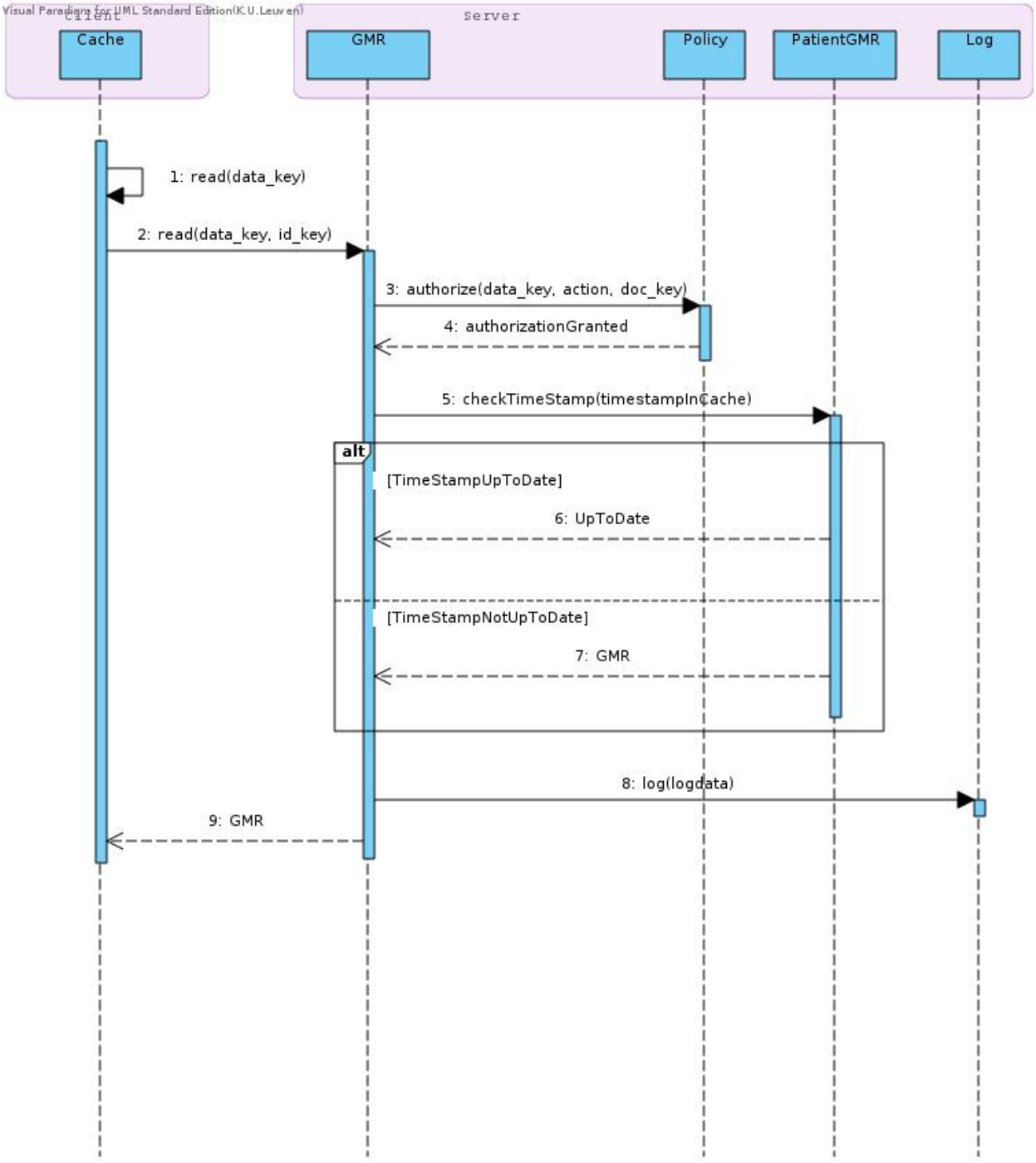

Figure 16: Interaction diagram: Retrieve GMR

10.4 Add prescription

Wanneer een dokter een voorschrift aan het dossier van een patient koppelt, wordt dit aangepast in het dossier van de patient zoals alle andere wijzigingen. Het verschil is dat er ook een kopie van het voorschrift

op de IDCard van de patient wordt gezet, waarmee deze naar de apotheek kan gaan.

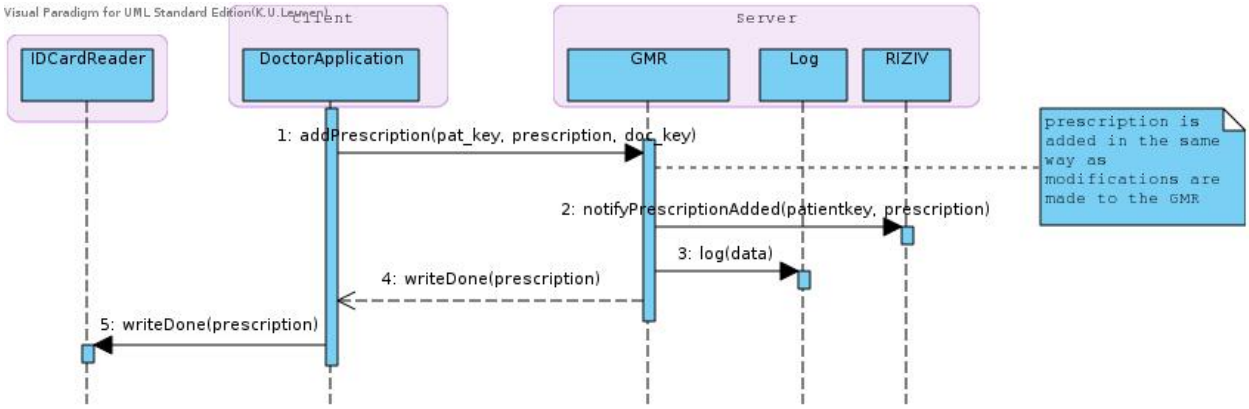

Figure 17: Interaction diagram: Add prescription

10.5 Validate prescription

Wanneer de patient naar de apotheek gaat om zijn medicijnen aan te kopen, wordt eerst het voorschrift van zijn IDCard gehaald. De apotheker overhandigd de benodigde medicijnen en duidt zowel op de IDCard als in het systeem aan dat het voorschrift werd gevalideerd. De boodschap naar het Ehealth platform is een boodschap die geen toegang tot het dossier vereist, enkel de key van de apotheker is belangrijk omdat geen andere persoon deze boodschap kan sturen.

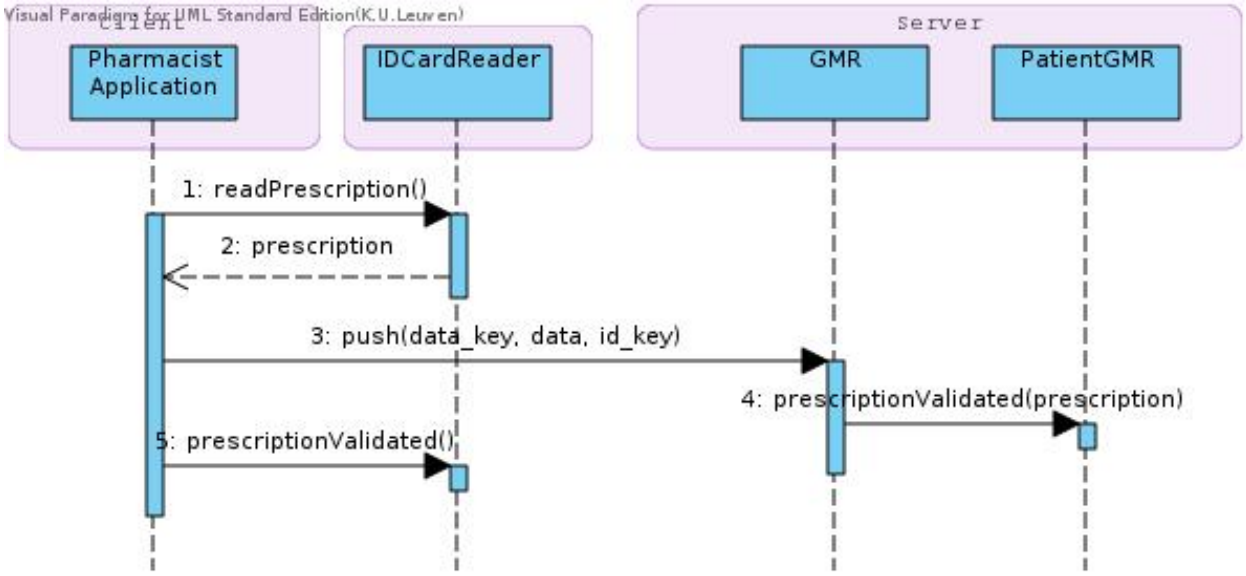

Figure 18: Interaction diagram: Validate prescription

10.6 Architecture background

Door de bovenvermelde sequentiediagrammen kunnen heel wat acties in het EHealth platform verklaard worden. Naast de voor de hand liggende acties (die direct door een sequentiediagram worden aangetoond), zijn ook de volgende scenario's verwezelijkt:

specialist zendt informatie naar de huisdokter: wanneer een patiënt een specialist consulteert, zal deze tijdelijke toegang krijgen tot het dossier van de patiënt. Hij kan hier dan zijn bevindingen in aanpassen en het dossier opslaan. Wanneer de huisdokter dan het dossier opnieuw opent, kan ook hij deze aanpassingen zien.

10.7 Related view packets

De interactie diagramma's zijn bedoelt om een aantal dingen duidelijk te maken, de architecturale beslissingen staan echter steeds beschreven in de andere views.