

Architecture of E-Health Flanders platform

Part I. Documentation Beyond Views

1 Documentation roadmap

1.1 Description of the parts

1.2 How stakeholders might use the package

1.2.1 Platform tester

1.2.2 Patient

1.2.3 Government

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

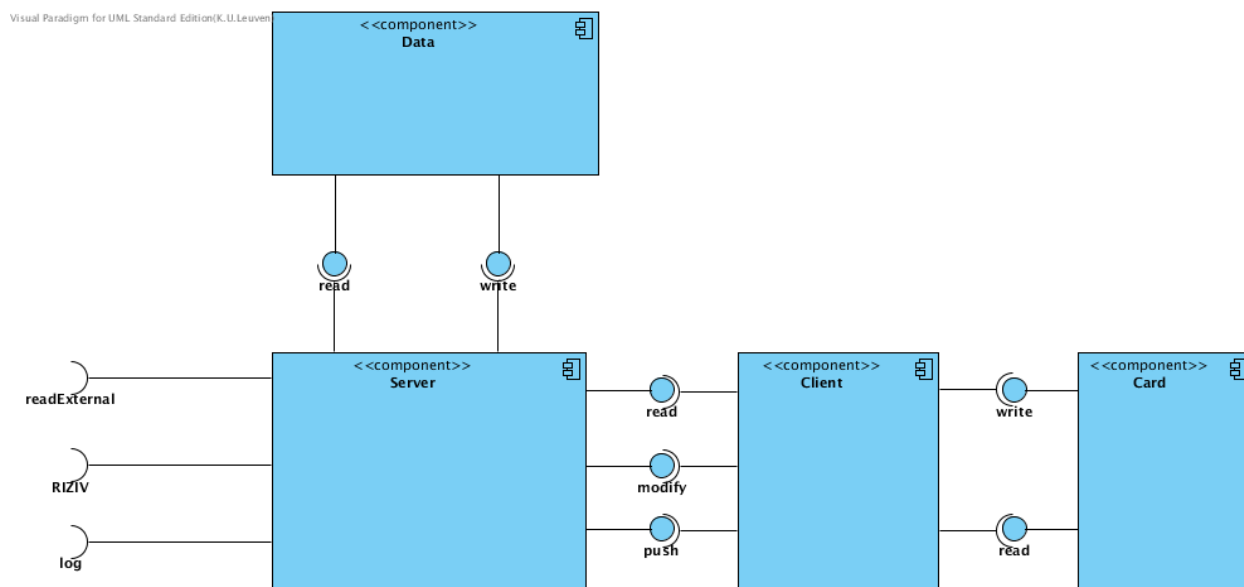
1.5.3 Assumptions

1.6 Other information

1.7 Related view packets

2 C&C Client and Server View: Overview

2.1 Primary presentation



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 Deployment diagram.

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:

2.3 Context diagram

TODO 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

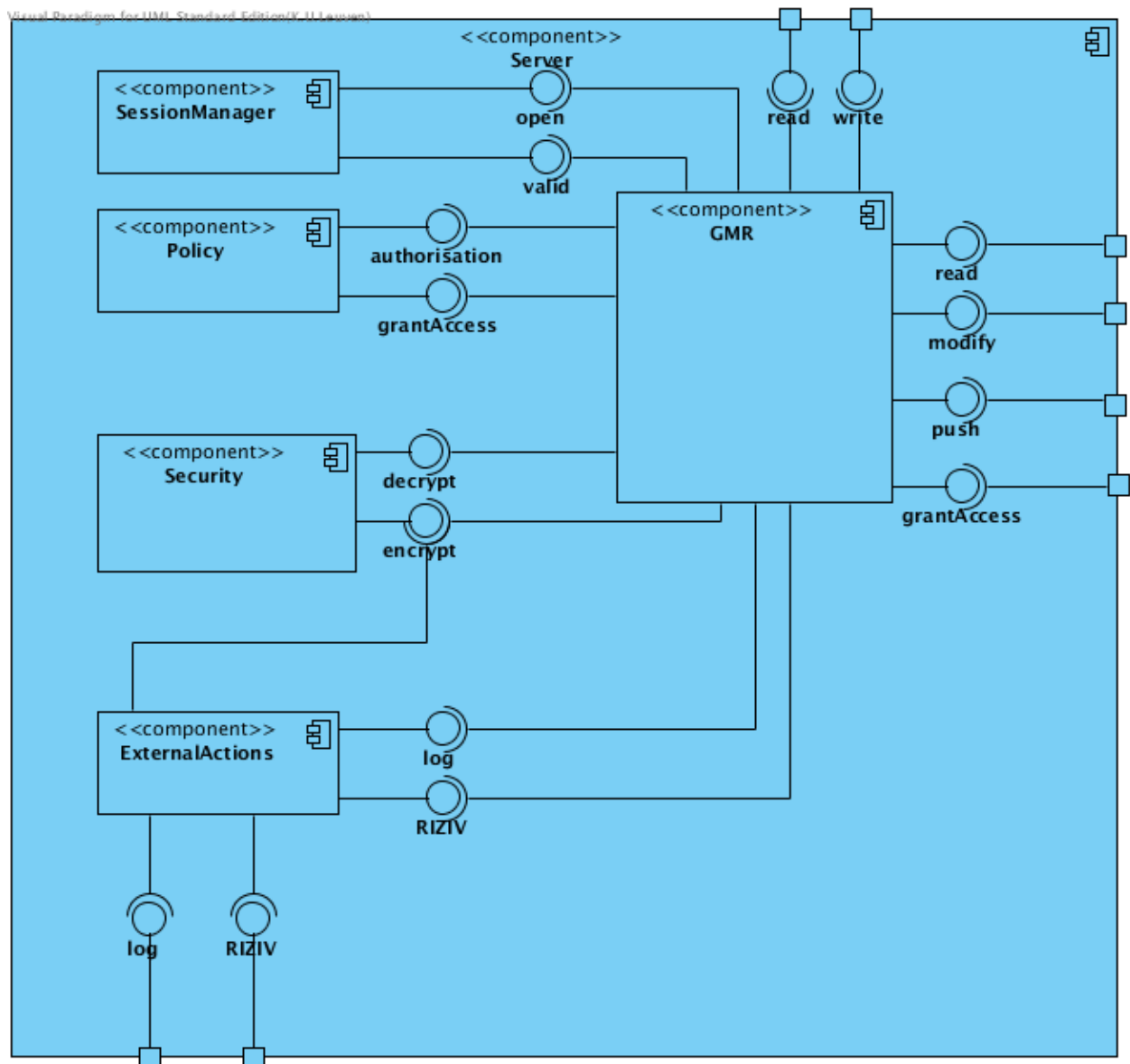
Enkele design beslissingen die hier te zien zijn is het gebruik van de card reader voor het identificeren van de gebruikers op de centrale database, meer hierover is te vinden in het ...TODO.

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.

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 teruggegeven 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 identificieren.

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 deployment 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 identificieren.

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 aanzienlijk 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 identificeren.

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 identificeert.

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 teruggegeven 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 identificeren.

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

authorisation Resources provided:

`authorize(data_key, action, id_key)`

De autorisatie 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 geautoriseerd.

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 identificeert.

Rationale and design issues:

zie ook deployment diagram 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 toegangsrechten 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 deployment diagram voor toegang tot de key database.

3.2.6 Element behavior

De policy component zal autorisatie toekennen aan bepaalde acties, als die persoon de actie mag uitvoeren. Indien dat niet mag zal de autorisatie geweigerd worden. Bepaalde personen hebben ook rechten om nieuwe rechten toe te voegen. Deze worden via `grantAccess` toegevoegd.

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 deployment diagram voor toegang tot de overheid.

RIZIV Resources provided:

`riziv(data)`

De data wordt verzonden naar de overheid toe. De component zorgt dat 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 deployment diagram 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 threshold 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]

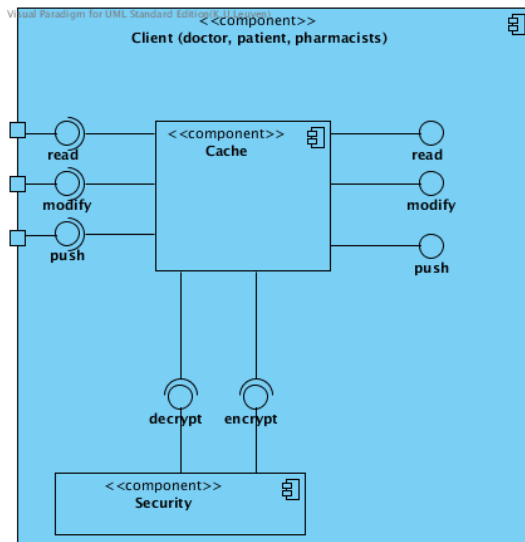
3.7 Related view packets

In het deployment view is te zien hoe de connect naar de externe servers tot stand komt.

In de interactie diagramma's zijn ook voorbeelden te vinden van hoe bepaalde interacties precies in hun werk gaan.

4 C&C Client and Server View: Client

4.1 Primary presentation



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.

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.

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**.

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]

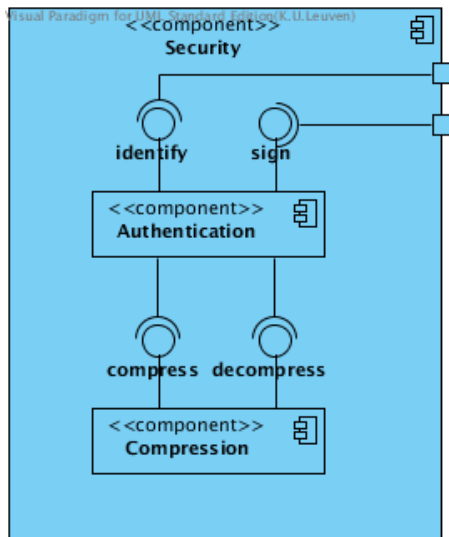
4.7 Related view packets

Deze client kant moet gezien worden in de volledige overview (zie 2 Client and Server Overview. Daarnaast is ook het deployment model belangrijk

5 C&C Client and Server View: Security

5.1 Primary presentation

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



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.

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.

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 gecomprimeerd zodat deze onleesbaar wordt voor iemand die deze niet kan decompresseren en zodat de grootte van de boodschap wordt verkleind.

Element requirements:

/

Rationale and design issues:

Door de boodschappen die worden verstuurd te comprimeren, wordt de boodschap zowel beveiligd, als verkleind, waardoor ook de verzendtijd wordt verkleind.

decompress Resources provided:

decompress()

Een boodschap die werd verstuurd en gecomprimeerd, moet eveneens worden gedecomprimeerd om leesbaar te zijn.

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

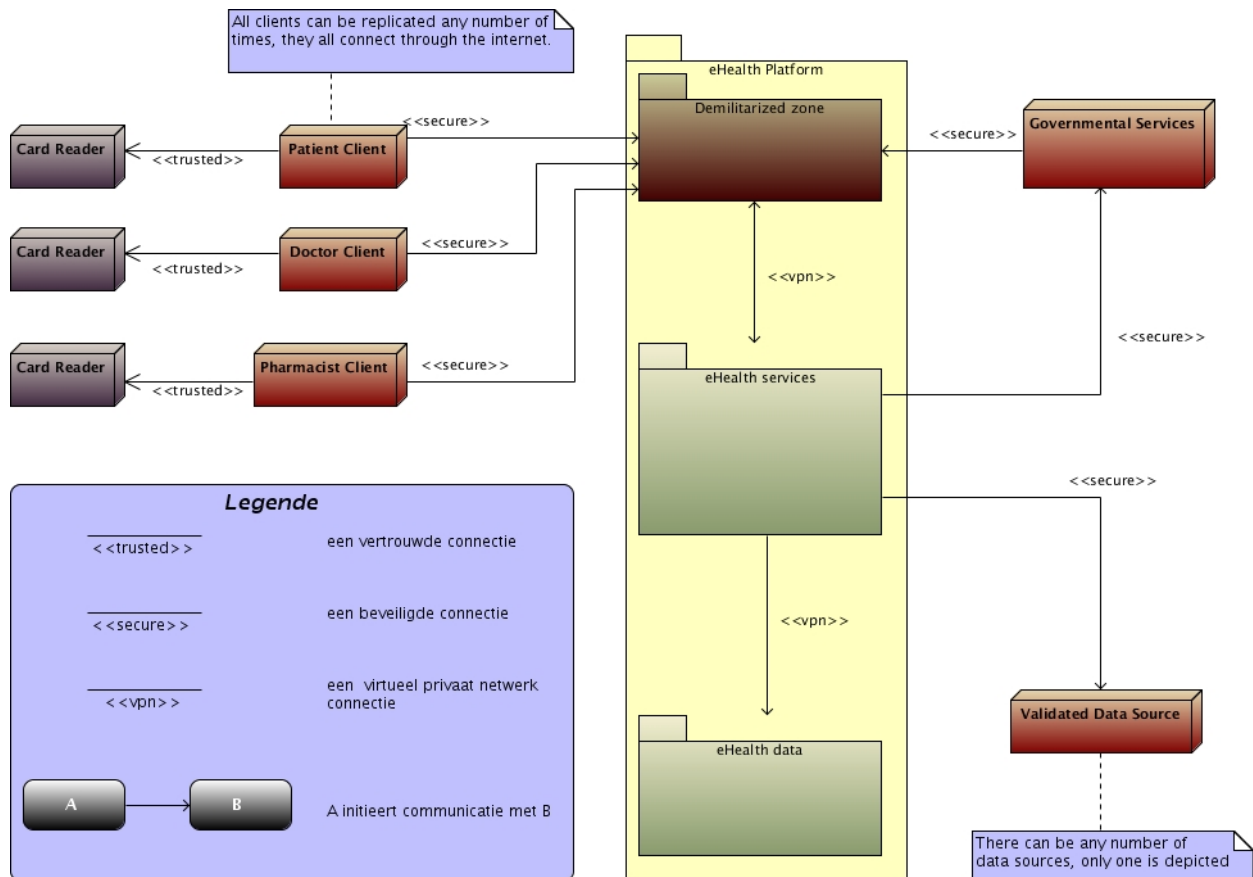


Fig. 1: deployment diagram giving a high-level overview of the system

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 autorisatie, 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 op 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

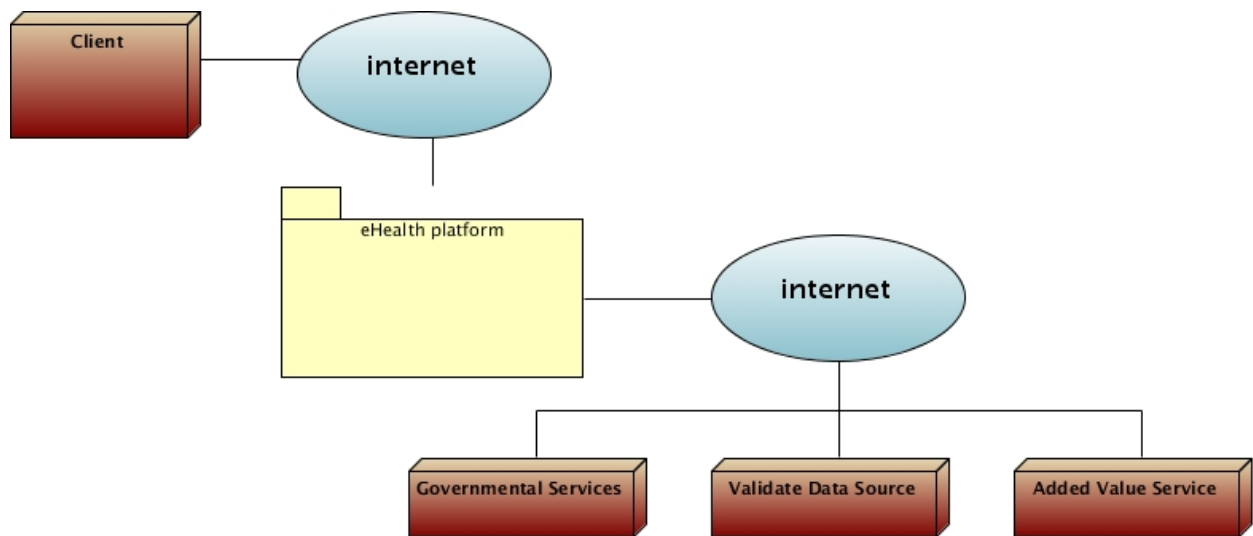


Fig. 2: Context diagram for the deployment diagram giving a high-level overview of the system

6.3 Context diagram

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 geautoriseerd 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

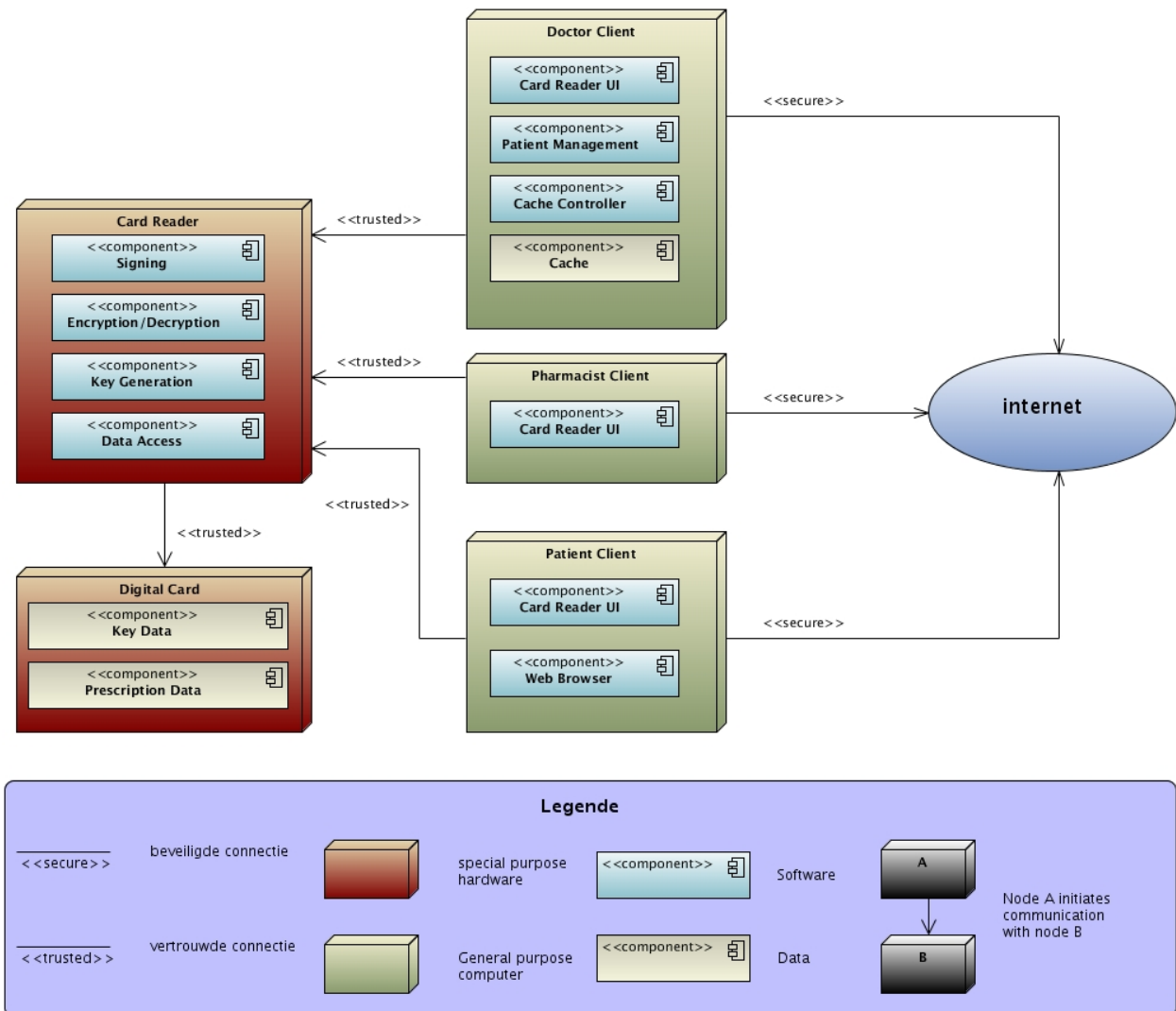


Fig. 3: Deployment diagram focused on the various clients in the system

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

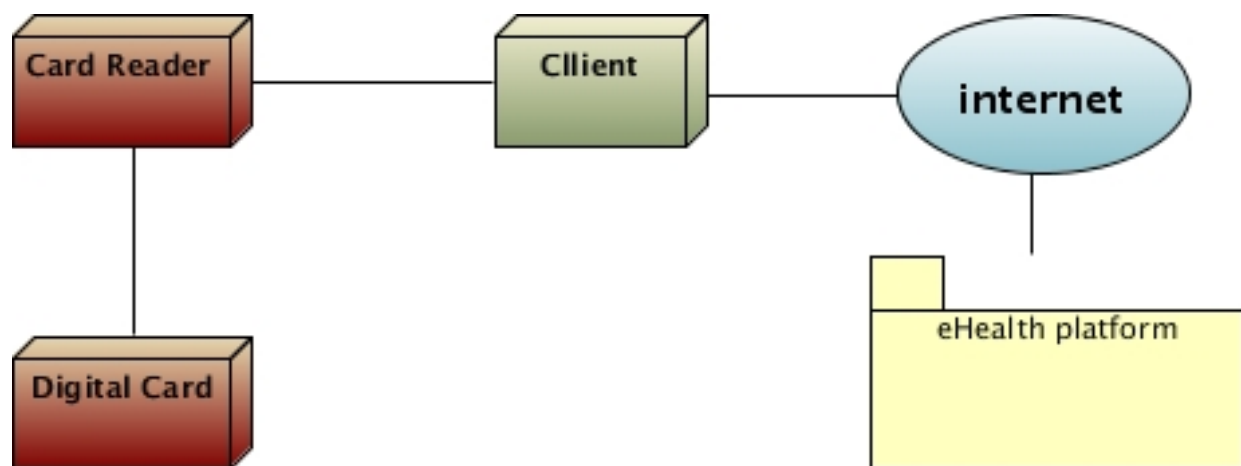


Fig. 4: Context diagram for the deployment view focused on the clients

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 falen. 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

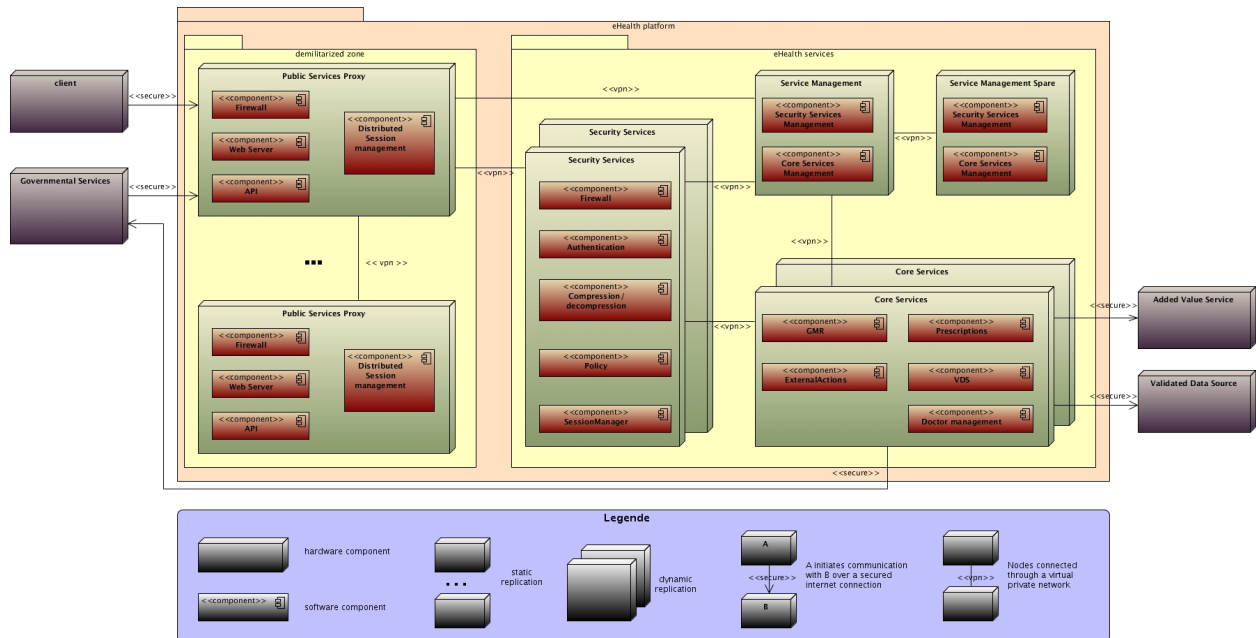


Fig. 5: Deployment diagram focused on the services in the platform

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 vormt 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 autorisatie toegelaten worden. Deze node zal dynamisch gerepliceerd worden, in sectie 8.5 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) **Compression/Decompression**

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 autorisatie 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) **ExternalActions**
Component die de link met de *Governmental Services* voorziet (loggen en RIZIV).
- (c) **Prescriptions**

8.2.2 Relations and their properties

8.2.3 Element interfaces

8.2.4 Element behavior

8.3 Context diagram

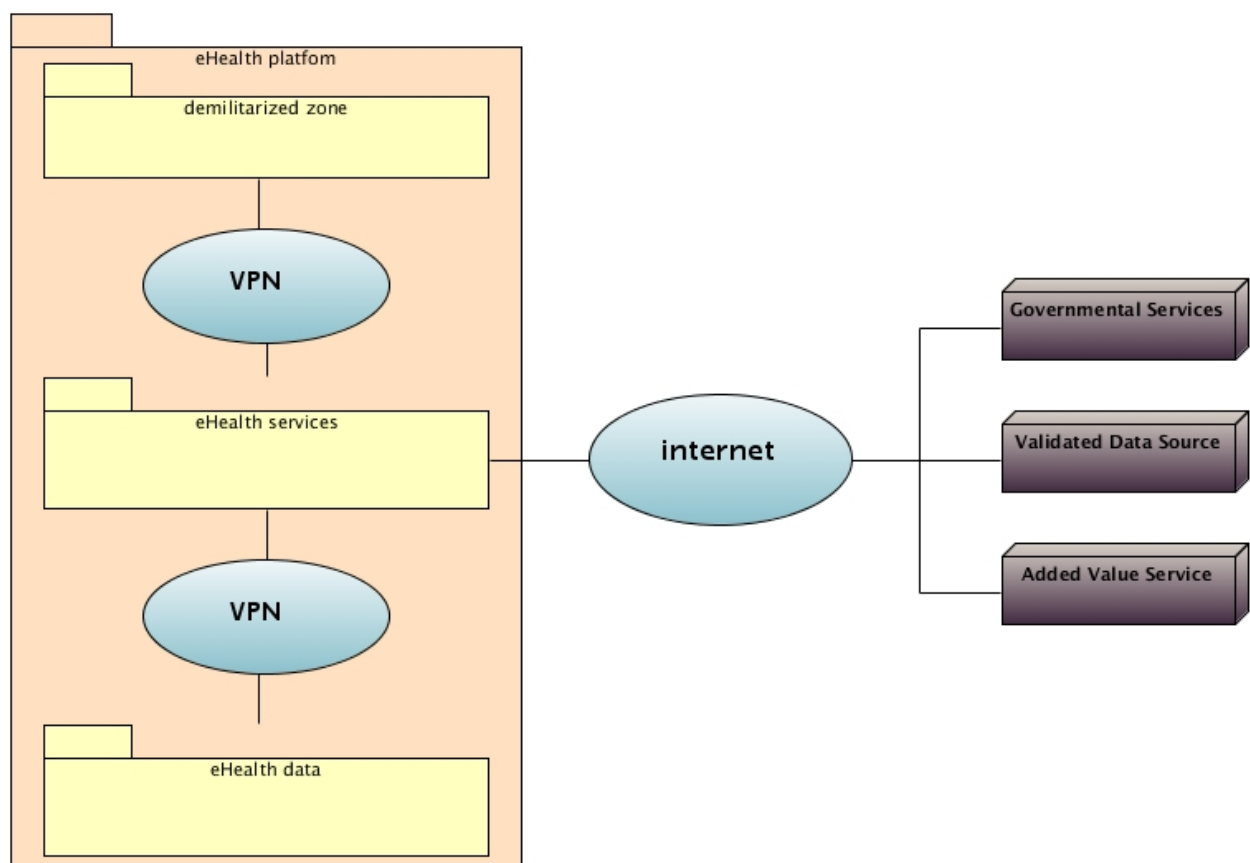


Fig. 6: Context diagram for the deployment diagram focused on the services in the platform

8.4 Variability guide

8.5 Architecture background

8.6 Other information

8.7 Related view packets

9 Deployment view data

9.1 Primary presentation

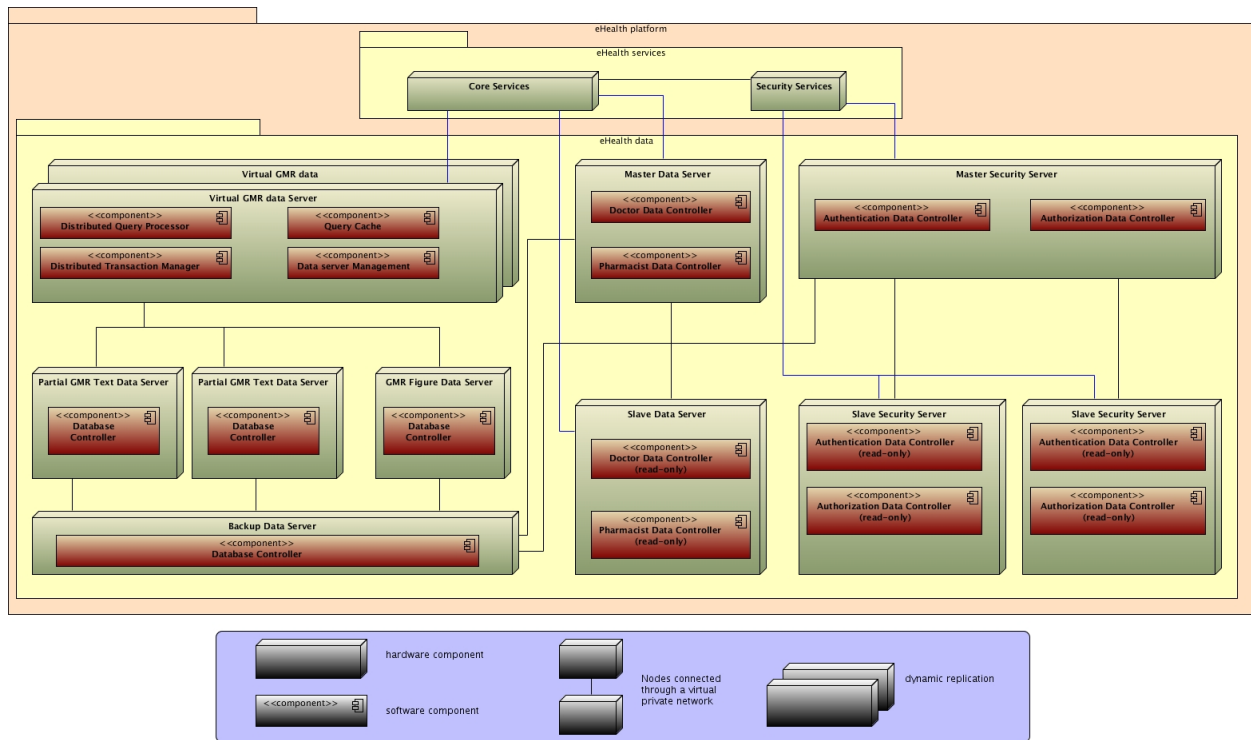


Fig. 7: Deployment diagram focused on the data storage in the platform

9.2 Element catalog

9.2.1 Elements and their properties

9.2.2 Relations and their properties

9.2.3 Element interfaces

9.2.4 Element behavior

9.3 Context diagram

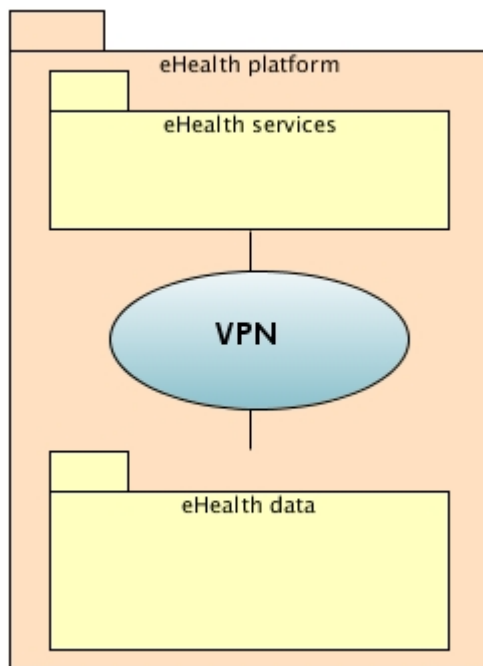


Fig. 8: Context diagram for the deployment diagram focused on the data storage in the platform

9.4 Variability guide

9.5 Architecture background

9.5.1 Rationale

9.5.2 Analysis results

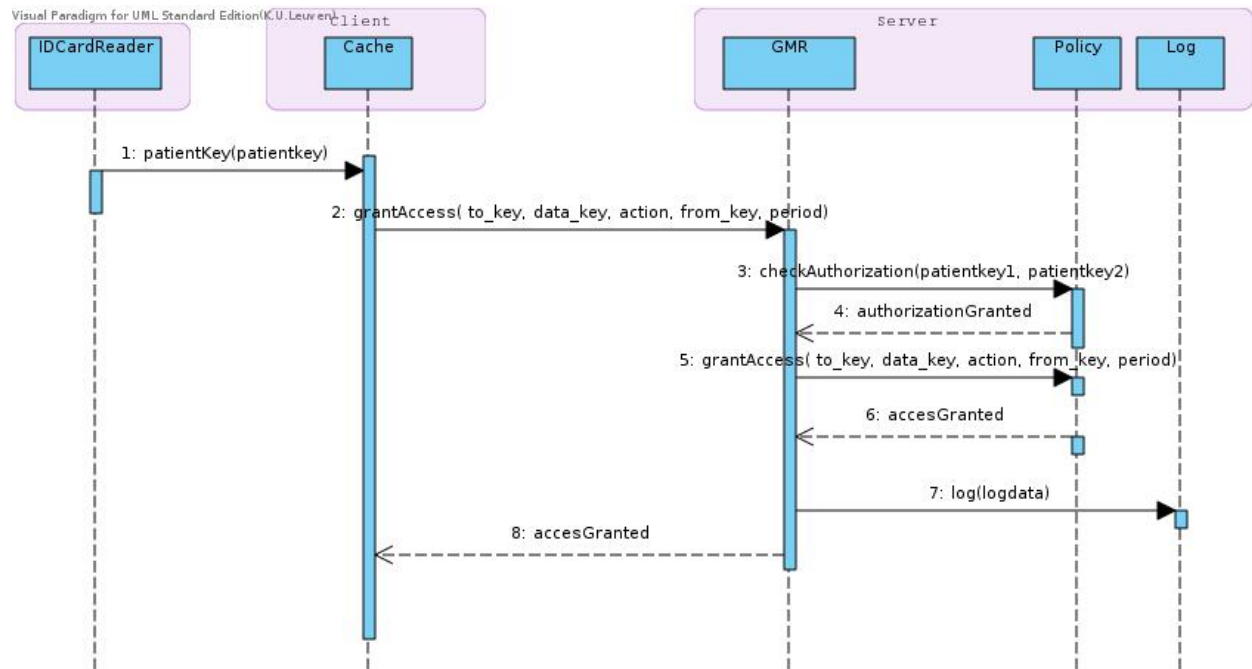
9.5.3 Assumptions

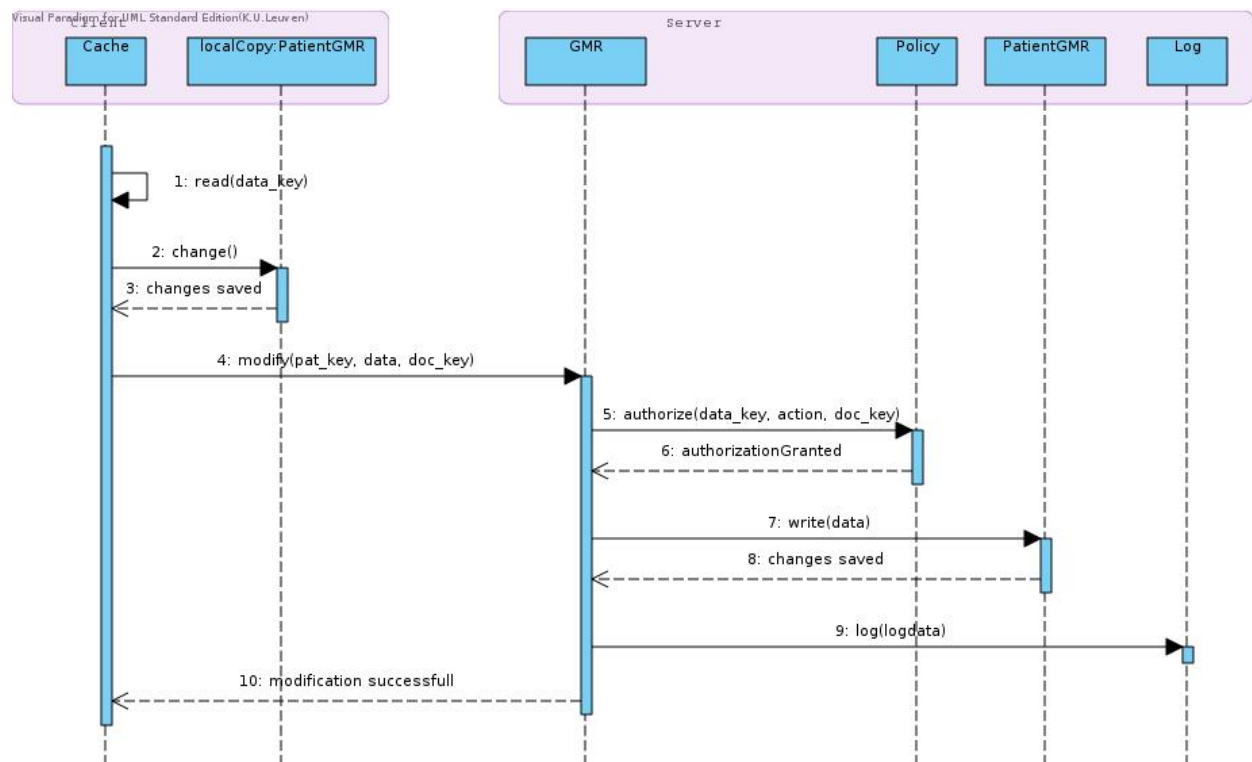
9.6 Other information

9.7 Related view packets

10 Interaction Diagrams

10.1 Primary presentation





10.2 Element catalog

10.2.1 Elements and their properties

10.2.2 Relations and their properties

10.2.3 Element interfaces

10.2.4 Element behavior

10.3 Context diagram

10.4 Variability guide

10.5 Architecture background

10.5.1 Rationale

10.5.2 Analysis results

10.5.3 Assumptions

10.6 Other information

10.7 Related view packets

