

EMERGO Deliverable 5.2: Technisch Ontwerp

Auteurs: Aad Slootmaker, Hub Kurvers

EMERGO penvoerende instelling:

Open Universiteit Nederland

- Onderwijstechnologisch Expertisecentrum
- Faculteit Psychologie
- Faculteit Natuurwetenschappen
- Ruud de Moor Centrum

EMERGO partner instellingen:

Universiteit Maastricht, Faculteit Psychologie

Radboud Universiteit, Faculteit Managementwetenschappen

Universiteit Utrecht, Faculteit Geowetenschappen

Datum: 7 september 2006

Versie 1.0

Kenmerk: U2006/6278

Onderwijstechnologisch expertisecentrum

OpenUniversiteitNederland

1. Inleiding	2
2. Technologische keuzes	3
3. Rollen	4
3.1. Rolfuncties.....	4
3.2. Beheerderrol.....	5
3.3. Ontwikkelaarrol.....	5
3.4. Docentrol.....	5
3.5. Studentrol.....	6
3.6. Rollentabel.....	6
4. Instituten, cursussen, casussen	6
4.1. Casussen.....	6
4.2. Casus- en casusrun-datamodel.....	7
4.3. Instituten en cursussen.....	9
5. Het datamodel	12
5.1. componentdata en componentstatus.....	12
5.2. Teksten.....	15
6. Componenten	19
6.1. Architectuur.....	20
6.2. Technische hoofdcomponenten.....	22
7. Tot slot	25

1. Inleiding

Dit document bevat het technisch ontwerp van de EMERGO-software. Het document gaat uit van de functioneel gewenste componenten en structuur van het programma, zoals die zijn beschreven in deliverable 1.3. Op technologisch gebied levert deliverable 5.1 de input. In dat document worden de keuzes beargumenteerd voor de te gebruiken software-pakketten en de te implementeren programma-architectuur.

Functioneel gezien is het begrip “casus” de kern van EMERGO. Een casus kunnen we beschouwen als een virtuele stage die een student moet doorlopen. De student krijgt een opdracht en een aantal hulpmiddelen om in een virtuele autonome wereld die opdracht naar behoren af te kunnen handelen. Een casus bestaat dus uit:

- een opdracht
- een virtuele wereld die:
- reageert op handelingen van de student
- zelfstandig (autonoom) van gedaante kan veranderen
- gereedschap dat kan worden gebruikt bij het afhandelen van de opdracht

De EMERGO-applicatie moet zowel constructie van een casus als het aanbieden van een casus aan studenten mogelijk maken (voor een nauwkeurige omschrijving zij verwezen naar het EMERGO Controlling Document). Dat betekent dat elke gebruiker in EMERGO een specifieke rol zal hebben. Zoals in deliverable 1.3 beschreven zal een gebruiker in een of meer van de volgende 3 gebieden actief zijn:

- het ontwikkelen van casussen
- het beheren van casussen
- het afspelen van casussen

Naast het gereedschap dat de student kan/moet gebruiken om een casus tot een goed einde te brengen, is er ook gereedschap dat een casusontwikkelaar nodig heeft om een casus te construeren, en gereedschap dat een beheerder kan gebruiken om casussen toegankelijk te maken voor studenten en docenten. Bovendien zijn er binnen EMERGO nog een aantal autonome onderdelen, die kunnen worden beschouwd als gereedschap dat niet direct zichtbaar is voor de gebruikers, maar dat wel nodig is om het doorlopen van een casus mogelijk te maken. We zullen in het vervolg naar al deze functionele eenheden verwijzen met de term “componenten”. Wanneer we het hebben over “gereedschap”, bedoelen we die componenten die een student ziet bij het doorlopen van een casus.

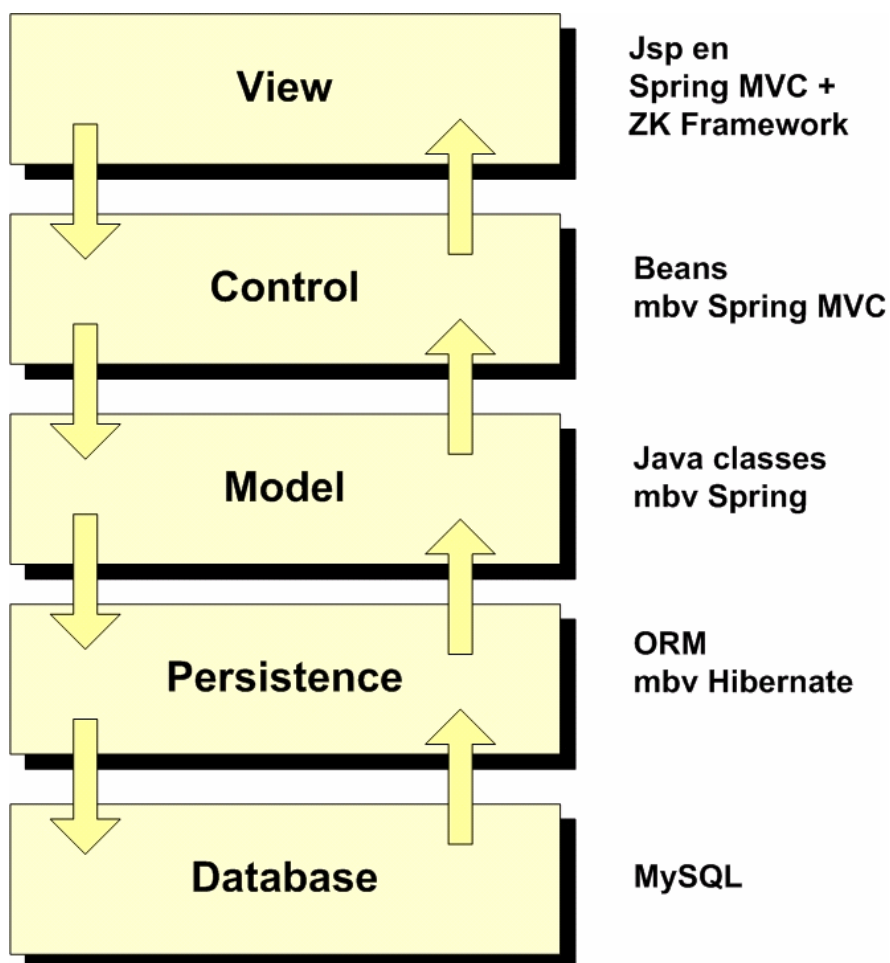
In de volgende hoofdstukken zullen we beschrijven hoe we de gewenste functionaliteiten technisch willen realiseren. We onderscheiden daarbij de drie belangrijkste onderdelen van het EMERGO-systeem: gebruikersrollen, casussen en componenten. De beschrijving zal niet alle details bevatten, omdat de gewenste functionaliteiten op het moment van schrijven ook nog nader moeten worden uitgewerkt, en omdat we ons nog verder moeten inwerken in de te gebruiken technologieën. Vooral het object- of domeinmodel zal nog verder uitgewerkt moeten worden, terwijl het datamodel al in grote lijnen vast ligt. Dit document zal in de toekomst nog de nodige aanvullingen behoeven.

2. Technologische keuzes

We hebben in deliverable 5.1 beschreven dat we voor EMERGO een Model-View-Controller-architectuur willen gebruiken, met een Web Services-laag (als onderdeel van de Controller-laag) voor de koppeling aan andere systemen, en verder willen we zo veel mogelijk gebruik maken van Open Source-pakketten. Meer gedetailleerd zal de lagenstructuur van EMERGO eruit gaan zien als getoond in figuur 1. Naast de 3 genoemde lagen zien we hierin ook nog de Database-laag, die via een extra laag, de Persistence-laag, gekoppeld is aan de Model-laag. De Persistence-laag kan gezien worden als de verbinding tussen het relationele datamodel en het objectgeoriënteerde domeinmodel.

Ook is in figuur 1 aangegeven hoe we de lagen technisch willen implementeren. De Open Source-wens leidt tot een keuze voor Java, MySQL, het Spring-framework voor implementatie van MVC, en voorlopig het ZK-framework of ECHO2 voor de userinterface. Voor de implementatie van de Persistence-laag hebben we nog geen definitieve keuze gemaakt. De belangrijkste kandidaten in combinatie met Spring zijn Hibernate, Ibatis, JDO-frameworks en Torque. Hibernate is de meest gebruikte, en zorgt voor een goede performance, Ibatis is minder populair dan Hibernate, maar even krachtig en een stuk eenvoudiger in het gebruik, Torque is een krachtige omgeving, die databeans genereert aan de hand van een configuratiebestand.

We willen in Eclipse gaan ontwikkelen, met gebruikmaking van de Spring-plugin.



Figuur 1: implementatie van de lagen

3. Rollen

In deliverable 1.3 zagen we dat de gebruikers van EMERGO een aantal rollen kunnen aannemen. De volgende rollen werden daar gedefinieerd:

- Casusontwikkelaar
- Ontwikkelteamlid
- Casusrunbeheerder
- Docent
- Student

Naast bovengenoemde rollen is er technisch gezien natuurlijk ook nog een EMERGO-beheerder. Deze beheerder kan gebruikers toegang verlenen tot EMERGO en hij zal de casusontwikkelaar-rollen moeten toewijzen. Ook kan hij ervoor zorgen dat nieuwe componenten aan het systeem worden toegevoegd.

3.1. Rolfuncties

De rol van een gebruiker in EMERGO bepaalt in hoeverre hij toegang heeft tot de functionaliteit van de verschillende componenten. Zo zal een casusontwikkelaar de inhoudelijke data van componenten moeten kunnen aanpassen. Een student mag de inhoud niet aanpassen, maar hij moet misschien wel gegevens selecteren. Een beheerder mag misschien helemaal geen rechten hebben om de component te starten.

In het verleden zijn de rollen binnen een casusontwikkelomgeving vaak meer gedetailleerd gedefinieerd. Per casus en per component is een rol dan te koppelen aan specifieke functionaliteiten. De rol van casusontwikkelaar is bijvoorbeeld onderverdeeld in een onderwijstechnoloog-, een inhoudsdeskundige- en een grafisch-vormgever-rol. Daarnaast kun je denken aan component-specifieke invoerders, bijvoorbeeld een gespreksontwikkelaar. Het voordeel van opsplitsen van een "hoofd"rol over verschillende "bij"rollen is dat we eenvoudig grotere projectteams een casus kunnen laten ontwikkelen zonder het risico te lopen dat teamleden elkaars data per ongeluk gaan overschrijven. Verder is het bij toevoeging van een nieuwe rol niet nodig de logica van een component aan te passen: een beheerder kan de rol via invoerschermen per component koppelen aan specifieke functionaliteiten, en het "aan" of "uit" staan van functionaliteiten wordt vervolgens per rol door het systeem doorgegeven aan de componenten.

Omdat we EMERGO technisch en wat betreft user interface liefst zo eenvoudig mogelijk willen houden, zullen we ons zoveel mogelijk beperken tot bovengenoemde rollen. In de componentlogica zullen we dan afhankelijk van de rol van de gebruiker verschillende userinterface-functionaliteit aanbieden. Wel willen we bij het ontwikkelen gebruikers niet alleen aan een casus koppelen, maar ook aan een component. Standaard zal elk casusteamlid toegang krijgen tot alle componenten, maar de casuseigenaar heeft de mogelijkheid de teamleden te verdelen over de componenten. Wanneer meerdere teamleden de data van een component kunnen aanpassen, moeten ze onderling afspreken wie wanneer wat gaat aanpassen.

Een gebruikersaccount dient voor toegang tot EMERGO gekoppeld te zijn aan tenminste één rol. Heeft een gebruiker meer dan 1 rol, dan zal hij na authenticatie moeten kiezen in welke rol hij de EMERGO-sessie gaat doorlopen.

De rol die een gebruiker heeft, bepaalt over welke componenten en functionaliteiten die gebruiker in potentie kan beschikken. Wat die gebruiker op een bepaald moment daadwerkelijk ziet op een EMERGO-scherm, is ook nog afhankelijk van andere elementen, met name van de status van gebruiker en componenten, en van de componentlogica-inhoud.

3.2. Beheerderrol

Bij installatie van EMERGO wordt automatisch een account toegevoegd met de beheerderrol. Dit account is nodig om in eerste instantie gebruikers te kunnen toevoegen, en om die gebruikers vervolgens een rol te geven. Hij zal ontwikkelaars moeten benoemen, en, per casus, docenten en studenten.

3.3. Ontwikkelaarrol

Voor een ontwikkelaar geldt het volgende:

- een ontwikkelaar kan een geheel nieuwe casus definiëren
- een ontwikkelaar kan een nieuwe casus maken van een uit de bibliotheek geselecteerde casus (templatecasus of volledige casus)
- als een ontwikkelaar een nieuwe casus definieert wordt hij automatisch de eigenaar van die casus
- als eigenaar ontwerpt de ontwikkelaar de structuur van de casus
- als eigenaar benoemt de ontwikkelaar andere ontwikkelaars tot teamleden
- een ontwikkelaar ziet bij starten van EMERGO een lijst met casussen waarvan hij eigenaar is en een lijst met casussen waarin hij teamlid is
- als teamlid kan een ontwikkelaar de inhoud van de verschillende componenten van de casus vullen
- als eigenaar bepaalt de ontwikkelaar tot welke componenten een teamlid toegang heeft
- als eigenaar van een casus kan de ontwikkelaar, wanneer de casus nog in ontwikkeling is, die casus verwijderen
- als eigenaar kan de ontwikkelaar een casus die in ontwikkeling is in de bibliotheek zetten als template of als volledige casus
- als eigenaar kan de ontwikkelaar een casus in de bibliotheek inactief (= onzichtbaar voor anderen) maken

3.4. Docentrol

Een casusdocent heeft de volgende kenmerken:

- hij kan "runs" maken van de casussen waarin hij docent is
- als hij een run maakt wordt hij automatisch eigenaar van die run
- als eigenaar selecteert hij bij elke run uit de casusaccounts de studenten en eventuele andere docenten die toegang krijgen
- als eigenaar kan hij een run, zolang deze nog niet gestart is, verwijderen
- als niet-eigenaar heeft een rundocent alleen toegang tot de run als de starttijd gepasseerd is
- tijdens een run kan de eigenaar de runeindtijd en de lijst met studenten die toegang hebben tot de run aanpassen
- een rundocent kan statusinformatie over de run opvragen

3.5. Studentrol

Voor een student geldt:

- hij ziet bij starten van EMERGO een lijst van lopende casusruns waaraan hij als student meedoet
- na keuze van een casusrun waarmee hij al bezig is geweest ziet de student die run in een toestand die is afgeleid van de toestand die hij heeft achtergelaten; de achtergelaten toestand is eventueel aangepast onder invloed van externe gebeurtenissen (ingrepen van het systeem, toevoegen van documenten van andere studenten, mails van de docent, etc.)

3.6. Rollentabel

Uit bovenstaande volgt, dat we van de eerder genoemde rollen de “casusrunbeheerder”-rol en de “ontwikkelteamlid”-rol niet expliciet in de rollentabel in de EMERGO-database hoeven op te nemen. In de volgende paragraaf zullen we nader ingaan op de benodigde account- en rollentabellen. We zullen dan ook zien dat de EMERGO-beheerderrol verder moet worden uitgewerkt, omdat we instituten en cursussen in het datamodel gaan opnemen.

4. Instituten, cursussen, casussen

EMERGO zal uiteindelijk een bibliotheek met casussen bevatten, waarvan er een aantal geschikt zijn om aan studenten aan te bieden. Om de ontwikkeling van voorbeeldcasussen en templates rendabel te maken, ligt het voor de hand om één EMERGO-server te gebruiken, die toegankelijk is voor alle instituten die casussen willen gebruiken. Ontwikkelaars van de verschillende instituten zijn op die manier in staat elkaars casussen over te nemen zonder daar import- en exportprocedures voor te moeten gebruiken. Verder kunnen, ook binnen een instituut, casussen betrekking hebben op geheel verschillende inhoudsdomeinen. In de praktijk hebben we meestal te maken met vier lagen: een casus is onderdeel van een cursus, de cursus wordt aangeboden door een faculteit, en een faculteit is gebonden aan een instituut. Het zal vaak zo zijn dat op een cursuswebsite een rechtstreekse koppeling naar elk van de EMERGO-casussen staat, maar om in het EMERGO-systeem de data overzichtelijk te houden, en om goede filtering van de data mogelijk te maken, hebben we besloten ook instituten en cursussen onderdeel te laten uitmaken van het EMERGO-systeem. Dat heeft consequenties voor het ontwerp van de database, zoals we straks zullen zien. Eerst gaan we het begrip “casus” in meer detail bekijken.

4.1. Casussen

Een casus, waarvan we de functionele definitie in de inleiding hebben gegeven, kunnen we technisch vertalen naar een verzameling van componenten en daaraan gekoppelde data, die door een “casusontwikkelaar” als eenheid wordt gedefinieerd. Een verduidelijking van het begrip krijgen we door verschillende typen van componenten te onderscheiden:

- de casusdefinitiecomponent (voor elke casus noodzakelijk); heeft minimaal als data:
 - een casusidentificatie
 - een lijst gebruikers die toegang hebben tot de casus: de eigenaren en teamleden, dat zijn allen casusontwikkelaars
 - inhoudelijke casusgegevens

- een lijst van componenten die actief zijn in de casus
- status (“onder constructie”/“volledig”/“template”/“verwijderd”/“op slot”)
- actieve gebruiker-interactiecomponenten met als data:
 - inhoudelijke gegevens
 - (eventueel conditie-afhankelijke) statusgegevens
 - userinterface-gegevens (achtergrondplaatjes, venstergrootte/~positie, etc)
 - Elke casus heeft een eigen set van interactiecomponenten. De casuseigenaar selecteert ze uit de lijst met alle in EMERGO geïmplementeerde componenten. Die lijst wordt beheerd door de EMERGO-beheerder. Alleen hij kan componenten aan de totale lijst toevoegen of ervoor zorgen dat ze niet meer selecteerbaar zijn.
- systeemcomponenten die bepalen en bijhouden in welke toestand de actieve componenten zich bevinden, afhankelijk van de gebruikersacties (voor alle casussen noodzakelijk)
 - Meer concreet moeten deze componenten activiteit van gebruikers en interactiecomponenten verwerken, en op grond daarvan eventueel andere componenten aansturen, en ze moeten de status van de casusrun per student beheren.

4.2. Casus- en casusrun-datamodel

Functioneel wordt, zoals in deliverable 1.3 is beschreven, onderscheid gemaakt tussen een casus en een casusrun. Het ligt voor de hand deze tweedeling over te nemen bij het opstellen van het datamodel dat als basis van EMERGO gaat dienen. De functionele eisen hebben de volgende consequenties voor het datamodel:

- als een ontwikkelaar aangeeft dat hij een nieuwe casus wil ontwikkelen wordt er aan de casus-datatable een record toegevoegd; naast velden voor bijvoorbeeld casustitel, casusomschrijving, en casusicoon, bevat de tabel ook een “status”-veld. De initiële status van de casus is: “Onder constructie”. De “Onder constructie”-casus is alleen selecteerbaar voor casusontwikkelaars die rechten hebben op deze casus. De casusontwikkelaar die begint met een nieuwe casus wordt automatisch eigenaar (veld in het casusrecord). Hij kan uit de gebruikerslijst met casusontwikkelaars naar believen teamleden selecteren. Eigenaren hebben het recht de status van de casus aan te passen (zetten van de casus in de bibliotheek, verwijderen van de casus), teamleden kunnen alleen de inhoud van de casusdata aanpassen. Initieel krijgen teamleden toegang tot alle componenten, maar desgewenst kan een eigenaar per component bepalen welke teamleden de component mogen vullen. Hiermee kan hij voorkomen dat teamleden elkaars inhoud gaan overschrijven
- als de ontwikkelaars tevreden zijn met hun casus kunnen ze hem beschikbaar stellen voor anderen (bibliotheek); de status van de casus wordt nu ofwel “template” (de ontwikkelaar vindt de casus nog niet geschikt om aan studenten aan te bieden), ofwel “volledig” (de casus kan in deze vorm aan studenten worden aangeboden)
- een casus in de bibliotheek kan door de eigenaar of een andere ontwikkelaar niet worden gewijzigd. Hij kan wel worden gebruikt als template voor een nieuwe casus, dat geldt zowel voor de “template”-casussen als voor de “volledige” casussen. Als een ontwikkelaar een bibliotheekcasus selecteert, maakt EMERGO een volledige kopie, zowel van de casusdefinitie als van de inhoud van de componenten, met dit verschil dat de gekopieerde casus de status

“onder constructie” krijgt, dat de gebruikerslijsten worden geleegd, en dat de ontwikkelaar tot nieuwe eigenaar wordt benoemd

- wat er gebeurt als een ontwikkelaar een casus verwijdert is afhankelijk van de status van de casus; als de casus “onder constructie” of “template” is, kunnen alle gegevens gerust uit de database worden verwijderd; een “volledige” casus kan echter in een casusrun gebruikt zijn. Dan hebben student- en docentstatusgegevens alleen betekenis als de casus nog aanwezig is. We kunnen afspreken dat we de casusdata niet uit de bibliotheek verwijderen, maar de status van een dergelijke casus op “verwijderd” zetten. De casus is niet meer zichtbaar in de bibliotheek, en komt ook niet meer in aanmerking als basis voor een nieuwe casusrun
- een docent kan bij starten van EMERGO kiezen voor monitoren van een casusrun (als er een lopende run is waarvoor hij docentrechten heeft), of construeren van een nieuwe casusrun. In het laatste geval krijgt hij de casusrunconstructiecomponent te zien. Hij kan uit de lijst van alle “volledige” casussen waarin hij een docentrol heeft, een casus kiezen die hij studenten wil aanbieden. Op dat moment wordt in de casusruntabel een record toegevoegd. De verhouding tussen casussen en casusruns is als volgt:

- bij elke casusrun hoort precies 1 casus
- bij elke casus behoren 0 of meer runs

Een casusrunrecord dient minstens de volgende velden te hebben:

- een casusrunidentificatie
 - een verwijzing naar de bijbehorende casusidentificatie
 - een verwijzing naar de eigenaar, dat het docentaccount dat de run definieert
 - een titel + omschrijving
 - status (“onder constructie”/“gearchiveerd”/“lopend”/“test”)
 - start- en eindtijd
- een casusrun heeft in eerste instantie de status “onder constructie”. Zodra de starttijd gepasseerd is, verandert de runconstructiecomponent de status naar “lopend”. Als ook de eindtijd gepasseerd is, verandert de status naar “gearchiveerd”. We moeten hierbij een controle toevoegen die erop let dat de eindtijd is ingevuld op het moment dat de starttijd gepasseerd wordt, en dat die eindtijd geldig is (dus later dan de starttijd). Als niet aan de voorwaarden voor correcte invoer van een casusrun voldaan wordt, blijft de status “onder constructie”
 - een casusrun kan worden verwijderd zolang zijn status “onder constructie” is. In dat geval worden alle gegevens uit de tabellen verwijderd
 - alle casusrungegevens zijn aanpasbaar zolang de runstatus “onder constructie” is
 - van een “lopende” casusrun is de eindtijd aanpasbaar, en is ook de gebruikerslijst aanpasbaar
 - wijzigen van de gegevens van een bibliotheekcasus of een lopende/gearchiveerde casusrun willen we zoveel mogelijk voorkomen (met uitzondering van de genoemde gegevens), om het risico van niet consistent zijn van statusgegevens uit te sluiten. We bieden in dat geval dan ook geen wijzigingsfunctionaliteit aan. In noodgevallen dient een EMERGO-applicatiebeheerder in werking te komen, die rechtstreeks toegang tot de database heeft
 - een bijzonder geval is de status “test”. Als een casusontwikkelaar aangeeft dat hij een casus die bij hem “onder constructie” is, als student wil doorlopen, zal de casusconstructiecomponent onder water een nieuwe run moeten aanmaken, met als gebruiker de ontwikkelaar. Het ligt voor de hand de ontwikkelaar een keuzemenu te geven, waarin hij kan aangeven welke rol hij wil aannemen, wat start- en eindtijd moeten zijn, en of hij wil doorgaan met een eventuele eerdere test, of opnieuw wil beginnen. Zolang de testrun draait,

mogen de gegevens in de bijbehorende "onder constructie"-casus niet gewijzigd worden, om problemen ten gevolge van inconsistente statusgegevens te vermijden. De status van de casus wordt op "op slot" gezet, zodat andere teamleden de gegevens niet kunnen aanpassen

4.3. *Instituten en cursussen*

We hebben besloten ook instituut- en cursusinformatie in het EMERGO-systeem op te nemen. Elke casus wordt gekoppeld aan een cursus, en elke cursus wordt gekoppeld aan een instituut. De casusontwikkelaars willen we wel per instituut groeperen, maar niet per cursus. Het zal immers vaak voorkomen dat een ontwikkelaar voor meer dan één cursus wordt ingezet. Bij de Open Universiteit bijvoorbeeld zijn dezelfde grafisch vormgevers en onderwijstechnologen betrokken bij een groot aantal cursussen. Van de andere kant hebben de casusrangebruikers vaak ook toegang tot alle andere casussen die bij een cursus behoren, maar niet tot casussen die bij andere cursussen van het instituut behoren. We willen dus twee soorten accounts onderscheiden in de EMERGO-database:

- de "institution accounts" zijn actief bij de constructie van casussen. De bijbehorende tabel kruist records uit de "accounts"-tabel met records uit de "institutions"-tabel. Elk institution account is aan een rol gekoppeld via een kruistabel tussen de institutionaccounts-tabel en de roles-tabel. Een institution account heeft ofwel de rol van casusontwikkelaar, ofwel de rol van administrator
- de "course accounts" zijn actief in de casusruns. De bijbehorende tabel kruist records uit de "accounts"-tabel met records uit de "institutioncourses"-tabel. Elk course account is aan een rol gekoppeld via een kruistabel tussen de courseaccounts-tabel en de roles-tabel. Een course account kan een student- of een docentrol hebben.

Verder spreken we af dat een gebruiker, als hij verbonden is aan meer dan een instituut, voor elk instituut een apart account krijgt.

4.3.1. Institution accounts: de EMERGO-beheerder

Bij installatie van EMERGO is de roles-tabel gevuld met standaardwaarden, en worden één accounts-record, één institutions-record, één institutionaccounts-record, en één institutionroles-record toegevoegd. Dit zijn de records die nodig zijn om de hoofd-EMERGO-beheerder te laten werken. Het accounts-record bevat de inlog-gegevens van de beheerder, het institutions-record heeft betrekking op een "dummy"-instituut. Het institutionroles-record koppelt het institutionaccounts-record aan de waarde "administrator" uit de roles-tabel. Als een gebruiker met dit account inlogt in EMERGO, dan heeft hij alle beheermogelijkheden tot zijn beschikking. In eerste instantie zal de hoofdbeheerder institutions-records en courses-records toevoegen. Cursussen zijn namelijk onafhankelijk van instituten. Ze hebben een eigen ingang in het Centraal Register Opleidingen Hoger Onderwijs (CROHO). De beheerder kan hun gegevens daaruit eventueel importeren, of hij voert ze handmatig in. Bij elk ingevoerd instituut kan hij vervolgens de institutionaccounts invoeren. In feite vult hij de accounts- en de contacts-tabel, terwijl het programma uitgaande van het geselecteerde instituut hiermee de institutionaccounts-tabel vult. Elk institutionaccount krijgt van de beheerder de rol van casusontwikkelaar ("developer") of instituutbeheerder ("administrator"). Dit wordt genoteerd in de institutionroles-tabel.

Een hoofdbeheerder zal ook moeten kunnen regelen welke componenten een ontwikkelaar kan kiezen voor zijn casus. Hij zal componenten dus toegankelijk of ontoegankelijk moeten kunnen maken, en uiteindelijk ook functionaliteit moeten krijgen om de componententabel uit te breiden.

4.3.2. Institution accounts: de instituutbeheerder

Een door de hoofdbeheerder benoemde administrator is voor het systeem een instituutbeheerder. Hij mag cursussen selecteren uit de CROHO-lijst, die aldus aan zijn instituut worden gekoppeld via een kruistabel ("institutioncourses"). Hij moet vervolgens voor elke cursus de docenten van die cursus en de studenten die zich voor die cursus hebben ingeschreven aan EMERGO toevoegen. Hij voert dus rechtstreeks de course accounts in. Het EMERGO-systeem voegt dan zelf de ingevoerde gegevens toe aan de accounts- en de contacts-tabel (als ze daar niet al in voorkwamen), en vult een record in de courseaccounts-kruistabel. Hij kan, om dubbele accounts voor een gebruiker zoveel mogelijk te voorkomen, bij invoeren van de course accounts ook selecteren uit de lijst met al ingevoerde accounts bij andere cursussen (niet noodzakelijk door hemzelf, wel van hetzelfde instituut natuurlijk).

4.3.3. Institution accounts: de casusontwikkelaar

Een casusontwikkelaar moet bij starten van EMERGO eerst een cursus selecteren waaraan hij gaat werken. Hij kan kiezen uit de al bij zijn instituut geregistreerde cursussen. Na keuze van de cursus krijgt de casusontwikkelaar een lijst met de "onder constructie"-casussen te zien (als die er zijn), plus de complete casusbibliotheek. In de casusbibliotheeklijst kan hij filteren op instituut en cursus.

Wanneer de ontwikkelaar aan een nieuwe casus gaat werken (van scratch, of gekopieerd uit de bibliotheek), wordt die casus automatisch gekoppeld aan de institutioncourses-kruistabel, via de kruistabel "coursecases". Vanaf dat moment is de nieuwe casus onwijzigbaar gebonden aan cursus en instituut. Een verwijzing naar de casusontwikkelaar wordt geplaatst in het eigenaar-veld van het cases-record, en wordt bovendien in het eerste "casedevelopers"-record voor deze casus gezet. Het is niet mogelijk om dezelfde casus in meerdere cursussen te gebruiken. Het datamodel laat dit wel toe, maar de functionaliteit bouwen we niet in in het domeinmodel; het kopiëren van een volledige casus naar een constructie-casus is namelijk functioneel hiermee vergelijkbaar.

De ontwikkelaar bepaalt vervolgens welke componenten deel uitmaken van de nieuwe casus. (De componenten zullen we meer gedetailleerd beschrijven in hoofdstuk 5.) Ook kan hij teamleden selecteren die hem gaan ondersteunen bij het invoeren van data in de componenten. De op instituut sorteerbare lijst waaruit hij kan selecteren bestaat uit alle gebruikers die voorkomen in de institutionaccounts-tabel en de casusontwikkelaar-rol hebben. Hij kan dus ook teamleden selecteren die bij andere instituten werken. De kruistabel "casedevelopers" tussen "institutionaccounts" en "cases" registreert wie teamlid is. Let wel, de kruistabel koppelt aan "cases" en niet aan "coursecases". Dit maakt het mogelijk om bij een nieuwe casus die gebaseerd is op deze casus, eenvoudig na te gaan wie aan de oorspronkelijke casus heeft meegewerkt. De nieuwe casuseigenaar kan de teamledenlijst (waaraan hij wel zelf is toegevoegd) eventueel ongewijzigd laten, zodat de leden zelf hun ingevoerde data waar nodig kunnen aanpassen.

Een casuseigenaar kan als metadata aan de casus ook "rollen" toevoegen. Deze "virtuele" casusrunrollen moeten we goed onderscheiden van de "echte" EMERGO-rollen. De EMERGO-rollen liggen vast in het systeem verankerd. Niemand kan deze rollen toevoegen, aanpassen of verwijderen. In de componentlogica wordt hard gecontroleerd op deze rollen. De rollen hebben voor alle casussen per component dezelfde functionaliteit. De casusrunrollen daarentegen zijn niet bekend bij de componentlogica. Ze zullen over het algemeen per casus verschillen. Ze spelen alleen tijdens een casusrun, ze worden dus gekoppeld aan de EMERGO-rollen docent of student, en ze beschikken in principe per component over dezelfde functionaliteit als de bijbehorende EMERGO-rol. Het verschil zit in de aan de casuscomponent gekoppelde data. Een casusontwikkelaar zal per component voor elke

runrol de inhoud moeten invoeren (uiteraard dient hij te kunnen beschikken over kopieerfunctionaliteit). Bovendien heeft hij de mogelijkheid het verloop van de casusrun rol-afhankelijk te maken. Dat wil zeggen dat hij condities kan opstellen waarin wordt getest op de runrol.

4.3.4. Course accounts: de docent als runeigenaar

Een docent-courseaccount ziet bij starten een lijst van casusruns waaraan hij als docent deelneemt, en een lijst met alle casussen waarvoor hij de rol van docent heeft gekregen. Uit de casussenlijst kan hij een casus selecteren om er een nieuwe run van te maken. Dit kan overigens ook een casus zijn waarvoor momenteel 1 of meerdere runs lopen. De casussen kunnen uit verschillende cursussen afkomstig zijn, maar de cursussen zijn wel alle van hetzelfde instituut.

Als de docent een nieuwe run gaat maken, wordt er een record toegevoegd aan de "runs"-tabel. Dit record bevat onder andere een verwijzing naar het bijbehorende record in de coursecases-tabel. Het docent-courseaccount wordt in een veld in het record toegevoegd, waardoor de applicatie de docent beschouwt als de runbeheerder. Deze runbeheerder bepaalt vervolgens welke bij deze cursus horende courseaccounts in de huidige run mogen meedoen. Deze informatie wordt opgeslagen in de "runaccounts"-tabel, dat is een kruistabel tussen courseaccounts en runs. Om samenwerken in groepen mogelijk te maken wordt de runaccounts-tabel door EMERGO omgezet in een door de eigenaar aanpasbare "rungroups"-tabel (zie 4.3.6). Deze rungroups vormen het uitgangspunt van een run. Dat wil zeggen dat een gebruiker de casus in een status ziet die gerelateerd is aan de rungroup waartoe hij behoort.

Wanneer er in een casus meerdere runrollen gedefinieerd zijn, zal de runeigenaar voor elke rungroup moeten aangeven welke runrol hij heeft.

4.3.5. Course accounts: student en docent als rundeelnemers

Als een gebruiker met een course account inlogt, zal hij een lijst krijgen van alle casusruns waarin hij actief is. In deze lijst hebben de runs waarvan hij niet de eigenaar is, de status "lopend". Na selectie van een van die runs kan het nog zijn dat hij voor die run in meerdere groepen gedefinieerd is. Hij zal dan eerst een keuze moeten maken voor een van de rungroups waarin hij voorkomt. De casustoestand die hij vervolgens te zien krijgt is afhankelijk van drie dingen: zijn EMERGO-rol, de status van de rungroup waartoe hij behoort, en de runrol van de rungroup, die de inhoud van de componenten bepaalt.

Door meerdere runrollen te definiëren heeft een ontwikkelaar de mogelijkheid van een casusrun een rollenspel te maken. Een student werkt dan bijvoorbeeld in een team van personen met elk een eigen specialisme aan de oplossing van een casus (psycholoog, verzorger, teamleider, therapeut, etc.). Elk teamlid wordt vertegenwoordigd door een andere student. Een eigenschap van een rollenspel is dat activiteiten van de ene runrol gevolgen kunnen hebben voor de casusstatus die andere runrollen zien. Een verzorger kan bijvoorbeeld een bericht sturen naar de projectleider. Als de verzorger op "verstuur" klikt moet iedere gebruiker in de projectleider-rol de status van de mail-component zien veranderen: de tekst van het bericht (plus andere informatie) is toegevoegd aan de rungroupcomponent-status voor alle projectleider-rungroups. En als een teamlid een rapportage van de patiënt gemaakt heeft, dan moet die rapportage voor alle andere teamleden in het patiëntdossier te vinden zijn.

4.3.6. Van course accounts naar run groups: samenwerken in groepen

De runaccounts bepalen dus welke courseaccounts in een run meedoen. In bepaalde situaties is het denkbaar dat studenten en/of docenten met elkaar samenwerken bij het afhandelen van een casus, en als het ware als één persoon aan de run deelnemen. Om dit mogelijk te maken gaat EMERGO bij het runnen van een casus niet uit van de runaccounts, maar van de "rungroups"-tabel. Deze werkwijze is overgenomen van het DU-project Espace. Het komt erop neer dat elk runaccount gekoppeld is aan een rungroup. Dit gebeurt middels de "rungroupaccounts"-tabel. Standaard krijgt elk runaccount zijn eigen rungroup, dus er is een 1-op-1-verbinding tussen deze 2 tabellen. Wil een docent dat studenten met elkaar samenwerken, dan moet hij de rungroupaccounts-tabel zo aanpassen dat meerdere runaccounts naar het zelfde rungroups-record verwijzen. Uiteraard hoeft hij niet zelf handmatig de tabellen aan te passen, maar hij krijgt een groep-constructie-gereedschap. Hij geeft een groep een naam, en selecteert de runaccounts die tot de groep moeten behoren. EMERGO past vervolgens de tabellen aan. We kunnen er bijvoorbeeld voor kiezen de docenten geanonimiseerd aan de run te laten deelnemen, door ze in een "docent"-groep te plaatsen. De kruistabel-constructie maakt het ook mogelijk dat een runaccount in theorie tot meerdere rungroups kan behoren (een student kan zo bijvoorbeeld in meerdere casusrunrollen aan de run deelnemen).

5. Het datamodel

Figuren 2 en 3 tonen een schematische weergave van het datamodel, respectievelijk voor de casussen en voor de casusruns. Hierbij is onder andere de koppeling tussen de runs-tabel in figuur 3 en de coursecases-tabel in figuur 2 niet weergegeven: het veld "coc_coc_id" in de runs-tabel is een verwijzing naar de coursecase-identificatie coc_id in de coursecases-tabel.

5.1. componentdata en componentstatus

In het datamodel van een component zien we dat de gegevens tot drie categorieën kunnen behoren:

- intrinsieke gegevens
- casusafhankelijke gegevens
- speeltijd-gebruikergegevens

5.1.1. Intrinsieke componentgegevens

De intrinsieke componentgegevens benoemen de component. Tot deze gegevens behoren bijvoorbeeld de naam van de component, een uitgebreide beschrijving, het type, het versienummer. Dit soort gegevens is onafhankelijk van de functionaliteit van de component. Voor alle componenten zullen dezelfde intrinsieke gegevensvelden aanwezig zijn. De "components"-tabel bevat alle noodzakelijke velden.

Een van de "components"-velden, het "com_com_id_parent"-veld, maakt het mogelijk een component aan een andere component te koppelen. Als een ontwikkelaar een component selecteert bij een nieuwe casus, dan controleert EMERGO of die component een parent-component heeft. Zo ja, dan wordt die parent-component automatisch ook aan de casus gekoppeld. Een componentbouwer, en eventueel ook de EMERGO-beheerder, kan op deze manier vastleggen dat een component bijvoorbeeld altijd in de PDA te zien is, of altijd als zelfstandig gereedschap in de locatiecomponent (in de locatiecomponent kunnen we dat gereedschap dan afhankelijk van de geselecteerde locatie wel of niet zichtbaar maken; dit is te regelen via casusspecifieke condities). Een ontwikkelaar hoeft dus niet expliciet eerst de PDA te kiezen en vervolgens welk gereedschap in de PDA zichtbaar is.

Met het veld "type" geven we aan of een component selecteerbaar is door de casusontwikkelaar, of niet-selecteerbaar. In het laatste geval gaat het dan bijvoorbeeld om systeemcomponenten die niet rechtstreeks zichtbaar zijn voor de gebruiker, maar waaraan wel data of status moet worden gekoppeld (en die dus in het datamodel moeten voorkomen). Ook een locatietool zal altijd noodzakelijk zijn. Selectie ervan is dus niet nodig. De aanwezigheid van een PDA is weer afhankelijk van de selectie van andere componenten, dus ook deze component hoeft een ontwikkelaar niet expliciet te kiezen.

5.1.2. Casusafhankelijke componentgegevens

De casusafhankelijke gegevens horen bij componenteigenschappen die, bij gebruik van een component in een casus, invariant zijn tijdens het spelen van die casus, maar die bij een andere casus een andere waarde kunnen hebben. Een casusontwikkelaar die een component selecteert voor gebruik binnen een casus moet voor die casus een aantal componentonafhankelijke gegevens invullen, en een aantal componentspecifieke gegevens.

De componentonafhankelijke gegevens bestaan momenteel uit de naam van de component en een lijst met de componentontwikkelaars. Een ontwikkelaar kan dus ervoor kiezen om de naam waarmee de component zich aan een casusrungebruiker presenteert te laten verschillen van de intrinsieke componentnaam. Deze voor alle componenten geldende gegevens staan in de "casecomponents"-tabel, of zijn daar rechtstreeks aan gekoppeld, zoals bijvoorbeeld de componentontwikkelaars middels de "componentdevelopers"-tabel.

Wanneer we het hebben over componentdata dan bedoelen we daarmee de componentspecifieke casusafhankelijke gegevens. We bewaren ze in de tabel "componentroledata", waaruit blijkt, zoals ook te zien in figuur 2, dat deze gegevens ook nog eens kunnen afhangen van de casusrunrol.

Voor de componentdata geldt niet dat ze, zoals de intrinsieke gegevens, in tabellen kunnen worden bewaard met per veld een data-element. De data die we moeten bewaren kunnen betrekking hebben op het uiterlijk van een component, op de aangeboden functionaliteit, en op de vakgebiedspecifieke inhoud. Zo kan het zijn dat we bij een component per casus willen laten instellen hoe groot het gebied is waarin video getoond wordt (bijvoorbeeld afhankelijk van de technische kwaliteit van de video-opnamen). Ook willen we het misschien mogelijk maken dat ontwikkelaars een eigen achtergrond toevoegen, of posities van knoppen en velden kunnen aanpassen. Ontwikkelaars moeten misschien ook de aanwezigheid van functionaliteit kunnen aanpassen, zoals de mogelijkheid tot printen, de mogelijkheid tot opvragen van de goede antwoorden in een toets, of de mogelijkheid tot maken van aantekeningen.

Wat als componentdata instelbaar is zal per component verschillen. Voor de vakinhoudelijke data is dat evident: de data kan voor elke component weer een heel eigen hiërarchie hebben. Maar ook op het gebied van de user interface en op het gebied van beschikbaar stellen van de functionaliteit zullen er grote verschillen te zien zijn. In de ene component kan het zinvol zijn om de grootte van het venster te kunnen instellen, in de andere niet; voor de ene component is een print-knop van belang, voor een andere heeft die knop geen betekenis. We zouden nu voor elke component een aparte tabel in het datamodel kunnen opnemen, met velden voor alle componentspecifieke data. Dat betekent dat als er een nieuwe component bijkomt, niet alleen het domeinmodel moet worden uitgebreid, maar ook het datamodel: we zullen tabellen moeten toevoegen. Dit maakt toevoegen of verwijderen van componenten complex.

We hebben voor een andere oplossing gekozen. De data van alle componenten brengen we onder in één tabel, de tabel "xmldata" (zie figuur 2). Sterker, we hebben in deze tabel slechts één veld nodig voor de data: het "data"-veld. We bewaren het geheel aan componentdata in dit veld in XML-formaat. In dit formaat kunnen we met name de hiërarchische structuren opslaan die benodigd zijn voor de vakinhoudelijke data. Dit soort data bestaat voornamelijk uit lijsten, met aan elk lijstitem weer een sublijst gekoppeld. Op deze manier zorgen we ervoor dat de data laag en de persistencelaag onafhankelijk zijn van de specifieke componentdata, en dus ook niet hoeven te worden aangepast bij toevoeging van een component.

Merk overigens op dat naar de xmldata-tabel verwezen wordt vanuit de "componentroledata"-tabel. Dit maakt het op een eenvoudige manier mogelijk om meerdere caserunrollen te koppelen aan gelijke componentinhoud (dit zal zelfs de standaard situatie zijn). In paragraaf 5.1.4 wordt nader ingegaan op het XML-formaat.

5.1.3. Speeltijd-gebruiker-componentgegevens

Tijdens het afspelen van een casus zal een component reageren op acties van de gebruiker of het programma, of op wijzigingen in de toestand van andere componenten. De actuele toestand van een component, in een casusrun, voor een gebruiker (rungleid), noemen we de componentstatus. Ook in dit geval zal elke component zijn eigen soort toestandgegevens hebben, zodat we geen universele status-tabel kunnen definiëren. Tenzij we natuurlijk weer, net als bij de componentdata, gebruik maken van het XML-formaat. Vooralsnog zullen we dan ook alle statusgegevens opslaan in de "rungleidcomponentstatus"-tabel, in het veld "xml_status". Dit veld bevat evenals het xml_data-veld tekst die is opgesteld in XML-formaat.

5.1.4. Het XML-formaat

Bij het tonen van de component in een run moeten we de componentdata en componentstatus met behulp van een XML-parser ontleden, om alle onderdelen van de component vorm te geven. Het XML-formaat maakt het mogelijk om boomstructuren te definiëren van property-value-combinaties. We zullen per component moeten definiëren wat de properties of tags zijn, zowel voor de datagegevens als voor de statusgegevens.

Figuren 4 en 5 tonen voorbeelden van respectievelijk een stuk xml_data-tekst en een stuk xml_status-tekst. In werkelijkheid zal met name de data-tekst nog wat complexer worden, omdat we er ook condities en vervolgacties (feedback, statusverandering, etc.) in willen opnemen. Merk op dat we niet de letterlijke teksten die zijn ingevoerd opnemen in de datatekst, maar wel verwijzingen naar records in de texts-tabel. Op deze manier kunnen we in verschillende data- en status-tekst velden naar hetzelfde tekstelement verwijzen, zonder dat we alle velden moeten controleren als de tekst wijzigt. Niet-tekst-informatie, zoals achtergrondafbeeldingen of introductievideo's, kan niet rechtstreeks in de velden worden opgenomen. We kunnen hiervoor het beste een aparte "BLOB"-tabel toevoegen (zie 5.2), waarnaar we rechtstreeks vanuit de XML-tekst verwijzen.

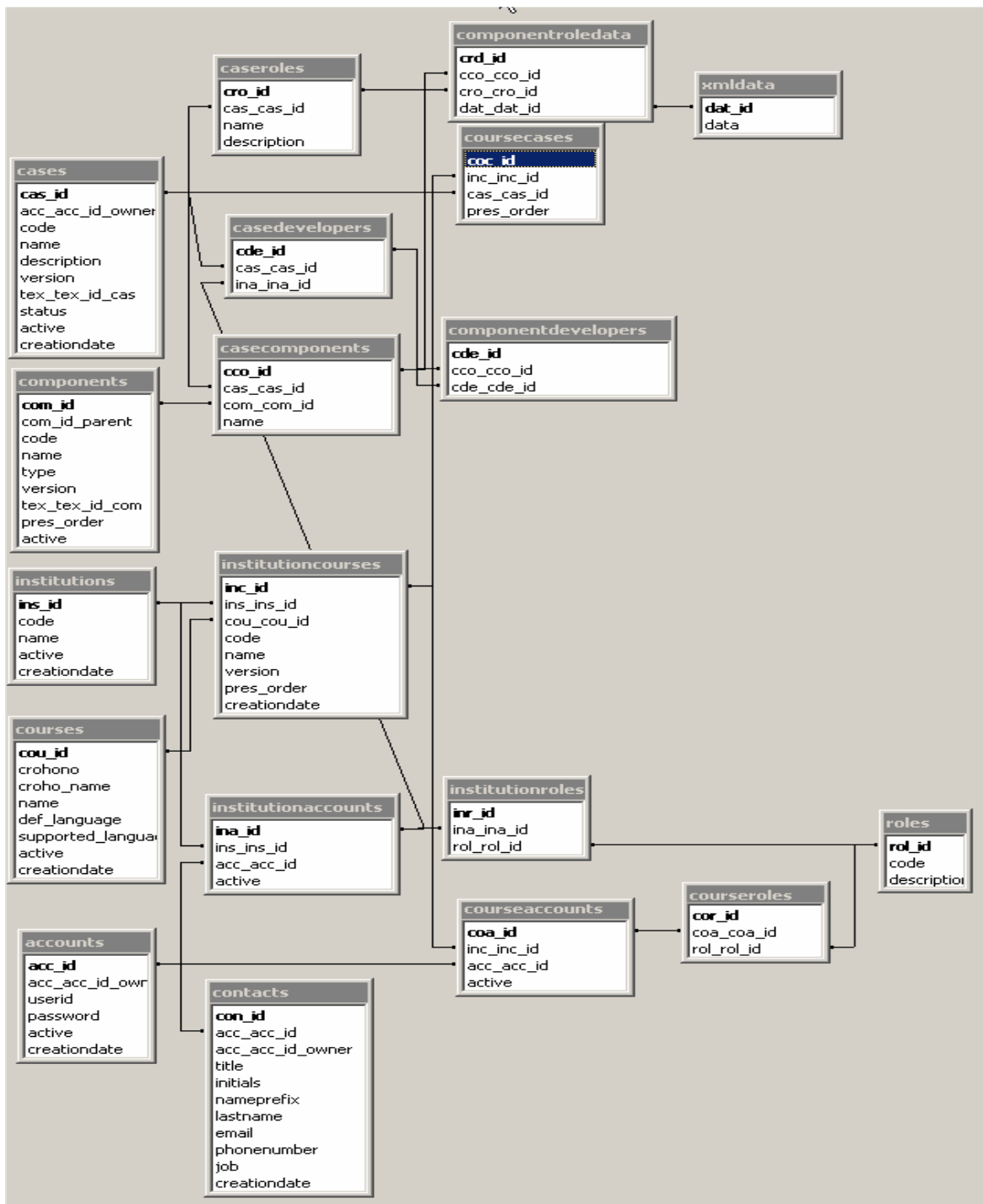
Via de script-tool voegt het programma conditie-teksten plus vervolgacties toe aan de XML-datavelden. Deze teksten bevatten over het algemeen nogal wat speciale karakters. Om problemen te voorkomen en eventuele leesbaarheid te vergroten zouden we de condities in een aparte "condities"-tabel kunnen onderbrengen, en in de XML-data alleen de betreffende condities-recordnummers kunnen opnemen.

5.2. Teksten

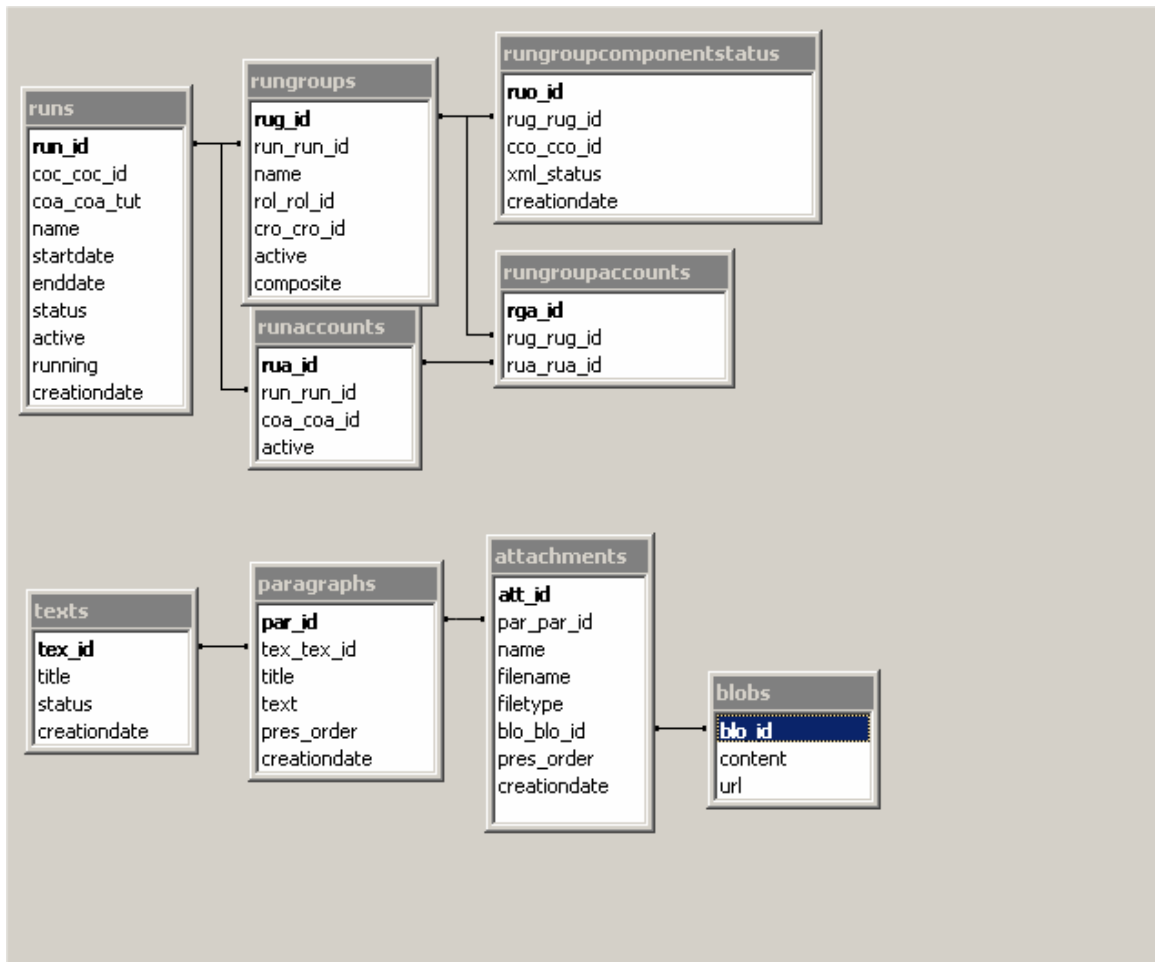
In figuur 3 is ook nog de tabel "texts" opgenomen. Via deze tabel kunnen uitgebreidere tekstfragmenten met bijlages en koppelingen naar externe websites worden toegevoegd. In de figuur zijn geen koppelingen aangegeven naar deze tabel. Maar kijken we naar de tabellen "cases" en "components" in figuur 2, dan zien we daarin "tex_tex_id_..."-velden, waarin een verwijzing komt naar een index in de texts-tabel. Via deze velden kunnen uitgebreidere beschrijvingen van een casus en van de componenten worden toegevoegd. Ook zagen we in 5.1 dat de componentdata en componentstatus verwijzen naar records in de texts-tabel.

De bijlagen bij de teksten bewaren we in de "attachments"-tabel. MySQL biedt de mogelijkheid om "Binary Large Object"-velden (BLOBs) te benoemen. Een BLOB is een object dat niet gebonden is aan een datatype, en "onbeperkt" is in grootte. Een veld van dit type is bij uitstek geschikt om eender welke bestanden in op te slaan. We willen dit type dan ook gebruiken voor het bewaren van de inhoud van de bijlagen. De attachments-tabel zoals te zien in figuur 3 beschikt nog over een veld "blo_blo_id" dat verwijst naar een record in de "blobs"-tabel. In die blobs-tabel kunnen we dan naast het blob-inhoudveld ook een "url"-veld opnemen, zodat er geen aparte "urls"-tabel nodig is. Het programma weet dan dat als het url-veld gevuld is een koppeling moet worden gemaakt, en dat als de blob gevuld is een geschikte blob-player (afhankelijk van het "filetype"-veld) moet worden gestart.

Op die manier kunnen we ook streaming video onderbrengen. Het betreffende blob-record heeft een leeg blob-inhoudveld, maar het url-veld is gevuld, en verwijst naar een bestand op een streaming-server.



Figuur 2: datamodel casussen



Figuur 3: datamodel casusruns

```

<background>
  <image>
    <BLOB id="2034"/>
  </image>
</background>

<list id="1">
  <label>
    <text id="34721"/>
  </label>
  <description>
    <text id="34932"/>
  </description>
  <content>
    <item>
      <text id="37123"/>
      <list id="2">
        <label>
          <text id="44721"/>
        </label>
        <description>
          <text id="44932"/>
        </description>
        <content>
          <item>
            <text id="47123"/>
            :
            :
            :
            :
          </item>
          :
          :
          :
          :
        </content>
      </list>
    </item>
    :
    :
    :
  </content>
</list>

```

Figuur 4: gedeelte uit de inhoud van een componentdata-veld

```

<list id="1" visible="true">
  <chosen>
    <text id="37123">
      <list id="2" visible="false">
        <chosen>
          <text id="44932"/>
          :
        </chosen >
      </list>
    </text>
  <text id="37124">
    <list id="2" visible="true">
      <chosen>
        <text id="44932"/>
        :
      </chosen >
    </list>
  </text>
  :
  :
  :
</chosen >
</list>

```

Figuur 5: gedeelte uit de inhoud van een componentstatus-veld

6. Componenten

Wanneer we het hebben over componenten moeten we de technische EMERGO-componenten onderscheiden van de functionele componenten zoals een EMERGO-gebruiker die ziet. Dat gebruikersgereedschap bestaat, zoals in de inleiding al vermeld, enerzijds uit middelen die in een casusrun ter beschikking staan voor studenten en docenten, en anderzijds uit middelen voor ontwikkelaars en administratoren om casussen en casusruns te definiëren en in te voeren.

De functionele componenten zullen niet per definitie één-op-één een equivalent hebben in de technische implementatie. Zo worden bijvoorbeeld in deliverable 1.3 een toetstool voor studenten en een toetsontwerptool voor ontwikkelaars beschreven. Technisch kunnen we dit implementeren als één component, waarvan de ontwikkelaar de invoerkant ziet, en de student de player-kant. Ook kunnen we componenten die veel op elkaar lijken technisch misschien vervangen door één component, die afhankelijk van de context waarin hij te zien is, een ander uiterlijk heeft. De in deliverable 1.3 beschreven selectieontwerptool kunnen we bijvoorbeeld onder andere combineren met de gespreksvoorbereidingstool. Ook bronnentool en dossiertool zouden hiervoor in aanmerking kunnen komen. Het komt er dan op neer dat een ontwikkelaar een dergelijke component 2 keer kiest bij een casus. Er zijn dan 2 casecomponents-records, die beide dezelfde "cas_cas_id" en "com_com_id" hebben, maar een verschillende waarde voor het "name"-veld. Uiteraard verschillen ook de "componentroledata"-waarden.

Naast componenten die veel op elkaar lijken zijn er ook onderdelen in componenten die in andere componenten ook nodig zijn. Deze onderdelen kunnen we als hulpcomponenten beschouwen, die te instantiëren zijn door de hoofdcomponenten. Elke instantiatie bezit dan een aantal kenmerken. Zo zou een videoplayer-hulpcomponent als data kunnen hebben: hoeveel instantiaties er momenteel zijn, en per instantiatie: de oudercomponent, zijn positie en grootte, de naam van het bestand dat hij moet tonen, welke besturingsknoppen beschikbaar zijn, etc. We kunnen deze data ook opnemen in de xml-data van de oudercomponent. Een record voor de hulpcomponent toevoegen aan de components-tabel is dan niet nodig.

Technisch gezien komt het er dus op neer dat we de componenten kunnen verdelen in componenten die in het datamodel zijn terug te vinden (en die dus kenmerken hebben die per casus en/of in een casusrun kunnen verschillen) en componenten die alleen zichtbaar zijn in het domeinmodel (maar waarvan casusafhankelijke data eventueel in de data van oudercomponenten zijn opgenomen).

Het programma bepaalt aan de hand van de waarde van het "type"-veld in de components-tabel welke componenten een ontwikkelaar kan kiezen. De componenten die gekozen zijn bij een casus moeten in een bepaalde volgorde worden gepresenteerd in een casusrun. Hiertoe vullen we het veld "pres_order" in diezelfde tabel in. Hoe hoger het nummer, hoe lager op de gereedschapslijst de component-startknop (of het ~icoon, of de ~hyperlink) zichtbaar wordt.

6.1. Architectuur

In deze paragraaf gaan we in op een mogelijke opzet voor de architectuur van EMERGO. Omdat we op het moment van schrijven nog bezig zijn met inwerken in Spring kunnen we nu nog geen goed beargumenteerde keuze maken. We zullen beginnen met het bouwen van de EMERGO-infrastructuur en vervolgens een eerste component implementeren. Aan de hand van onze bevindingen hierbij zullen we de architectuur nader uitwerken. Het volgende dient enkel als eerste aanzet te worden beschouwd.

In figuur 1 is te zien dat de EMERGO-componenten in een lagenstructuur worden gebouwd. In de database zijn alle variabele kenmerken van een component opgeslagen. Functies in de Persistencelaag geven de kenmerken door aan de eigenlijke componentfunctionaliteit, in de Modellaag. De visualisatie van de component vindt plaats in de Viewlaag, vanuit de Modellaag aangestuurd door de Controllerlaag.

6.1.1. Nieuwe componenten onafhankelijk van Datalaag

Een van de wensen die we hebben voor wat betreft de architectuur van EMERGO is dat we gemakkelijk componenten kunnen toevoegen. Deze wens wordt op het eerste gezicht bemoeilijkt door het gebruik van verschillende hiërarchische lagen. Het lijkt erop dat we bij toevoeging van een component in alle lagen aanpassingen moeten doen. Echter, door de keuzen die we gemaakt hebben op het gebied van dataopslag zal dit in de praktijk wel kunnen meevallen. We zagen in hoofdstuk 5 dat de structuur van het datamodel componentonafhankelijk is. Toevoegen van een nieuwe component betekent dat een beheerder een nieuw record moet toevoegen aan de components-tabel. Componentafhankelijkheid speelt pas een rol in de structuur van de inhoud van het xml-data-veld. Die inhoud wordt bepaald door de functionaliteit van de Modellaag. De Persistencelaag bevat alleen functionaliteit voor het vullen van tabellen en het ophalen van data uit tabelrecords, en is dus ook componentonafhankelijk. Met andere woorden, wanneer we een component toevoegen kunnen we de Databaselaag (afgezien van het toevoegen van een components-record) en de Persistencelaag ongemoeid laten.

6.1.2. Nieuwe componenten onafhankelijk van Viewlaag?

Als we naar de Viewlaag kijken, dan is een component te beschouwen als een venster waarin allerlei min of meer interactieve deelcomponenten zichtbaar zijn. Deze deelcomponenten, dit zijn de bovengenoemde hulpcomponenten, zullen in het algemeen in meerdere componenten een rol spelen. De deelcomponenten hebben altijd dezelfde functionaliteit, alleen hun inhoud en uiterlijk zullen per component verschillen. Die inhoud en uiterlijk worden gevuld via de Controllerlaag, vanuit de Modellaag van de hoofdcomponent, die ze weer ophaalt uit de xml-data (als ze door een vormgever zijn aan te passen), of uitgaat van constante waarden (als ze casusafhankelijk zijn).

We kunnen misschien voor de Viewlaag een algemene "component.jsp" bedenken. Deze wordt via de Controllerlaag aangestuurd door de Modellaag-functionaliteit van elke component afzonderlijk. De component.jsp bouwt het componentvenster op door de benodigde hulpcomponenten te tonen en te vullen met de doorgekregen waarden. Voor zover een nieuwe component gebruik maakt van bekende hulpcomponenten zullen we de Viewlaag dan niet hoeven aan te passen. Alleen als de component userinterface-functionaliteit nodig heeft die niet door de bestaande hulpcomponenten wordt gedekt moeten we een of meer nieuwe hulpcomponenten toevoegen (in Modellaag, Controllerlaag en Viewlaag), evenals functionaliteit in de algemene component om de nieuwe hulpcomponent te plaatsen en te vullen.

6.1.3. Algemene component in Modellaag

Een nieuwe component zouden we op die manier kunnen definiëren als een combinatie van bestaande hulpcomponenten. In de beschrijving moet dan komen hoe de hulpcomponenten (functioneel en op het gebied van userinterface) aan elkaar gerelateerd zijn, en welke eigenschappen ze elk afzonderlijk hebben. Zo'n beschrijving zouden we ook weer in XML-formaat kunnen realiseren. In de modellaag zou dan functionaliteit aanwezig moeten zijn om een component-XML-bestand (eventueel uit de database) in te lezen en te verwerken tot een functionele component.

6.1.4. Hulpcomponenten

Als we voor de algemene component-aanpak kiezen zullen we op de eerste plaats een zo compleet mogelijk overzicht van de hulpcomponenten moeten hebben. Afgaande op de tot dusver gewenste functionaliteiten komen de volgende hulpcomponenten in aanmerking:

- vensters
 - stijl popup of child; casusontwikkelaar kan grootte en positie instellen, al dan niet met rand en systeemmenu; achtergrond is te wijzigen (afbeelding); in casusrun zijn grootte en positie instelbaar afhankelijk van opties die ontwikkelaar heeft toegestaan
- deelvensters (~DIV's)
 - om een venster op te delen; de deelgebieden kunnen de hierna genoemde componenten bevatten; positie, afmetingen en achtergrond instelbaar door ontwikkelaar
- een boomuitklapstructuur-generator
 - kan de hierna genoemde hulpcomponenten bevatten; basispositie en breedte zijn instelbaar door ontwikkelaar
- tekstselectievelden
 - single-select, multiselect, mogelijk meerregelig per element, richtext-formaat; inhoud is gevuld door casusontwikkelaar, evenals positie en grootte; selectie is casusrun-status

- teksttoonvelden
 - richttext-formaat (kan ook afbeeldingen bevatten); door ontwikkelaar instelbare positie en afmetingen
- tekstbewarevelden
 - richttext-editor; door ontwikkelaar instelbare positie en grootte; kan tevens aangeboden functionaliteit van de editor instellen
- knoppen en links
 - eventueel gegroepeerd: radiobuttons, checkboxes, pushbuttons, matrixknoppen; ontwikkelaar kan positie, grootte, vorm (buttongraphic), caption instellen; actieparameters instelbaar; link bevat richttext (eventueel afbeelding); linkadres instelbaar
- mediaplayers
 - afspelen van (streaming) media (audio/video); besturingscontroles instelbaar door ontwikkelaar; positie en grootte (eventueel sizeable) instelbaar, eventueel in casusrun
- versleepgebieden (drag & drop)
 - een schuifgebied bestaat uit deelvensters; elk deelvenster (DIV) kan gevuld zijn met 1 of meer sleep-objecten; een sleep-object is een richttext-toonveld (inhoud/grootte/positie/overlapbaarheid/sleepactieparameterwaarde instelbaar door ontwikkelaar); na verslepen stuurt deelcomponent een melding met deelvenster-identificatie waarboven losgelaten wordt, plus huidige positie, plus sleepactieparameterwaarde; in casusrun wordt toestand per student bijgehouden

6.1.5. Interface-afhandeling

Het verloop van gebruikersactiviteiten is als volgt. Een gebruiker doet iets met/in een hulpcomponent. Vervolgens komt er een "submit" naar de Controllerlaag-functionaliteit van de algemene component. Deze roept aan de hand van een ingestelde actieparameterwaarde ("buttonDown", "buttonClick", "selected", "startDragging", "stopDragging", etc.) een van de standaardfuncties aan uit de Modellaag van de bijbehorende hoofdcomponent. Die standaardfunctie ("uh_Notify", overgeorven van de algemene component) activeert eerst een functie uit de Persistencelaag om de XML-data op te halen voor deze component. Vervolgens gaat een XML-parser (in een tussenlaag tussen Persistencelaag en Modellaag) het XML-data-gedeelte extraheren dat hoort bij de hulpcomponent, actieparameterwaarde en eventuele statuswaarde. Dit datagedeelte bestaat uit een lijst met conditie-identificaties, per conditie-identificatie gekoppeld aan een actielijst.

6.2. Technische hoofdcomponenten

We zullen nu een beschrijving geven van alle technisch te onderscheiden EMERGO-componenten. Hierbij geven we aan welke gegevens als casuscomponentdata moeten worden opgeslagen, welke gegevens tijdens de run voor elke student als status moeten worden bewaard, welke gegevens andere componenten kunnen opvragen of instellen, en welke gegevens we tijdens een runsessie tijdelijk moeten opslaan. De beschreven componenten komen grotendeels overeen met de in deliverable 1.3 genoemde functionele componenten.

Deze lijst zullen we gedurende de implementatie stelselmatig aanvullen en verbeteren, omdat de functionaliteit in eerste instantie nog niet op detailniveau is uitgewerkt, en omdat de specifieke eigenschappen van en relaties tussen componenten vaak pas tijdens de bouw duidelijk worden.

6.2.1. bronnentool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.2. takentool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.3. mail(ontwerp)tool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.4. agendatool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.5. lijstselectietool/gespreksvoorbereidingstool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.6. testtool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.7. communicatietool/interviewtool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.8. eventtool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.9. toetstool/itembanktool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.10. dossiertool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.11. portfoliotool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.12. docenttool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.13. studenttool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.14. scripttool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

6.2.15. casusrunstatustool

- omschrijving:
- casusafhankelijke data:
- rungroupafhankelijke data:
- sessiegegevens:
- benodigde hulpcomponenten:

7. Tot slot

Over een aantal technische aspecten hebben we op dit moment nog te weinig gegevens om er zinvolle uitspraken over te kunnen doen. Die aspecten zullen aan de orde komen tijdens het bouwen. We zullen ze hier alvast noemen.

- Communicatie tussen componenten
- conditie/scripttool
- Interface
- Domeinmodel
- Multilanguage
- Status
- Gebruik van echte mail
- Import/export
- Hosting en deployment