Gelaagde architecturen

Deel 1: Uitleg, terminologie en methoden

Lagenmodellen vormen een belangrijk onderdeel van de software architectuur van veel informatiesystemen. Een lagenmodel is er op gericht om de software zo te structureren, dat wordt voldaan aan bepaalde kwaliteitseisen, zoals een goede onderhoudbaarheid of vervangbaarheid. Lagenmodellen zijn abstracte modellen, die gemakkelijk verkeerd gekozen, begrepen en toegepast kunnen worden. Dit wordt mede in de hand gewerkt doordat belangrijke termen zeer verschillend worden uitgelegd.

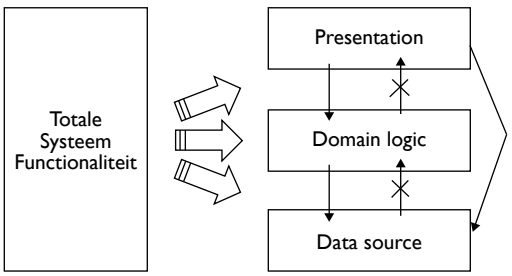
Dit artikel belicht de kenmerken van een lagenmodel en het onderscheid tussen logische en fysieke modellen, alsmede het nut van lagenmodellen en enkele problemen bij het gebruik ervan. Verder worden de soorten functionaliteit die in een laag kunnen voorkomen duidelijk geïdentificeerd en gedefinieerd. Deze basis wordt gebruikt in een aantal vervolgartikelen, waar methoden worden beschreven om tot logische en fysieke lagenmodellen te komen.

Wat is een lagenmodel?

Een lagenmodel schrijft voor hoe de totale functionaliteit van een informatiesysteem moet worden gestructureerd. Daarbij moet duidelijk worden gemaakt:

- Welke typen logica (functionaliteit of verantwoordelijkheid) onderkend moeten worden en hoe die typen gegroepeerd moeten worden tot lagen;
- Wat het hiërarchische niveau van iedere laag is. Daarbij geldt dat laag A boven laag B wordt gesteld, indien laag A elementen bevat die gebruik maken van functionaliteit van laag B;
- Welke communicatieregels gevolgd moeten worden. Die regels moeten duidelijk maken hoe de lagen onderling mogen communiceren, c.q. gebruik van elkaar mogen maken;
- Welke kwaliteitsdoelen kunnen worden gerealiseerd met dit lagenmodel. En hoe die doelen met deze lagen en communicatieregels bereikt kunnen worden.

Door een lagenmodel volledig, dus op deze manier, te specificeren, maakt de architect die het model opstelt zijn werk bespreekbaar en controleerbaar. En de ontwikkelaar die met het lagenmodel aan de slag gaat, krijgt voldoende informatie mee om belangrijke beslissingen te maken tijdens ontwerp en bouw.



Figuur 1: Een lagenmodel met 'The Tree Principal Layers'.

Zo is in figuur 1 een veelgebruikt lagenmodel te zien, waarbij de totale systeemfunctionaliteit is opgesplitst in 'The Tree Principal Layers'. Presentation logic, Domain logic en Data source logic zijn onderscheiden en als hiërarchische lagen in het model geplaatst. En de pijlen geven de communicatieregels weer: alleen aanroepen van boven naar beneden zijn toegestaan, waarbij een laag mag worden overgeslagen. Fowler geeft een redelijk volledige beschrijving van dit lagenmodel.

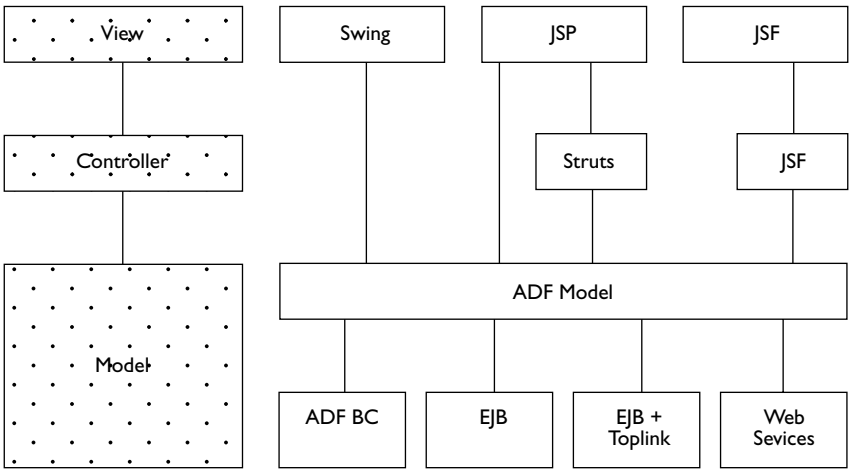
Waarom worden lagenmodellen gebruikt?

Een lagenmodel wordt vaak gekozen om de kwaliteit van de opgeleverde software te verhogen. Formeler geformuleerd: Een lagenmodel kan als maatregel worden gekozen om te voldoen aan een of meerdere niet-functionele eisen (kwaliteitseisen) die aan het informatiesysteem worden gesteld. Voorbeelden van veelgenoemde voordelen van lagenmodellen zijn:

- Hergebruik van functionaliteit in een lager gelegen laag, ten gunste van de juistheid en de uitbreidbaarheid;
- Logische opbouw van het systeem, ten gunste van de analyseerbaarheid, wijzigbaarheid en testbaarheid;

- Wijzigingen in een laag werken niet of nauwelijks door in andere lagen, ten gunste van de onderhoudbaarheid en de portabiliteit/ aanpasbaarheid;
- Vervangbaarheid van (een deel) van de functionaliteit van een laag of de infrastructuur, ten gunste van de portabiliteit.

In de bovenstaande voorbeelden zijn de volgende kwaliteitsattributen te herkennen (conform de terminologie van de ISO 9126 standaard): accuracy, maintainability (analysability, changeability, testability), portability (adaptibility, replaceability)
Lagenmodellen kunnen ook nadelen opleveren. In vervolgartikelen zal nader op de voor- en nadelen worden in gegaan.



Figuur 2: Een fysiek lagenmodel behorend bij het ADF-framework van Oracle.

Logische en fysieke lagenmodellen

In de literatuur over lagenmodellen wordt vaak geen onderscheid gemaakt tussen logische en fysieke lagenmodellen. Dat is jammer, want daardoor vallen soms zeer ongelijksoortige modellen allemaal onder dezelfde noemer 'lagenmodel'. De 'geest' van het onderscheid tussen logische en fysieke architectuurmodellen, zoals in TOGAF 9 is uitgewerkt op het gebied van componentmodellen, is hier overgenomen en op lagenmodellen toegepast.
Het logische lagenmodel richt zich vooral op de vraag: Hoe moet de totale systeemfunctionaliteit in lagen opgedeeld worden om aan welke niet-functionele eisen te voldoen? Kenmerken zijn: logische ordening, denken vanuit de functionaliteit/ verantwoordelijkheid, abstractie.
Het fysieke lagenmodel richt zich vooral op de vraag: Welke lagen moeten er, rekening houdend met de techniek, onderscheiden worden en hoe moeten die lagen gerealiseerd worden om aan welke niet-functionele eisen te voldoen? Kenmerken zijn: technisch werkingsmechanisme, denken vanuit de mogelijkheden van de ontwikkelomgeving en infrastructuur.

Onderstaande voorbeelden laten zien dat de twee typen fors kunnen verschillen. Figuur 1 toont een verdeling in 'The Tree Principal Layers' en is een voorbeeld van een logisch lagenmodel. Het belangrijkste voordeel van dit lagenmodel is dat 'domain logic' gecentraliseerd wordt en herbruikbaar wordt gemaakt voor verschillende 'presentations'. Waardoor aanpassingen en uitbreidingen relatief snel, goedkoop en accuraat kunnen plaatsvinden.
Een voorbeeld van een fysiek lagenmodel is de op het MVC-pattern gebaseerde architectuur van het ADF-framework van Oracle (figuur 2). Er zijn opvallende verschillen ten opzichte van een logisch lagenmodel. Rechts staan geen functionaliteiten, maar technologieën. Het model laat in de eerste plaats een werkingsmechanisme zien en de mogelijkheden van het framework om verschillende presentatie- en service technologieën te koppelen. Maar een doel als 'hergebruik van domain logic' wordt er zonder aanvullende regels niet mee bereikt.

Problemen met lagenmodellen

Een lagenmodel schrijft voor hoe een systeem moet worden gestructureerd. Helaas is goed ordenen en structureren moeilijk, foutgevoelig en tijdrovend. Daardoor bestaan er lagenmodellen waarvan bij nadere beschouwing onduidelijk is waarvoor ze dienen, wat nu precies de regels zijn en of het eigenlijk wel een lagenmodel is. En zelfs als het lagenmodel klopt, dan is het opvallend hoe gemakkelijk er tijdens ontwerp of bouw een fout met het lagenmodel gemaakt kan worden. Het is dus moeilijk om de voordelen, waarvoor het lagenmodel is opgesteld, ook echt te bereiken. Dit komt onder andere doordat:

- Het ontwerpen of kiezen van een lagenmodel geen eenvoudig werk is en heuristieken ontbreken;
- Eenvoudig een 'standaard' lagenmodel gekozen kan worden, dat onvoldoende past bij de eisen of gekozen technologie;
- De terminologie met betrekking tot lagenmodellen niet eenduidig is en soms ronduit tegenstrijdig;
- Ontwikkelaars aan de slag moeten met een onvoldoende gedefinieerd lagenmodel;
- Ontwikkelaars aan de slag gaan met onvoldoende kennis en inzicht.

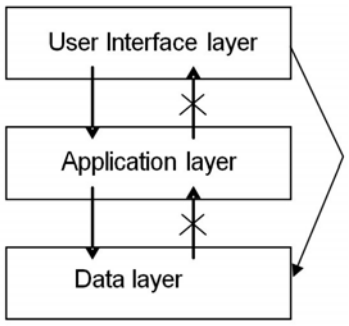
Problemen met de definitie

Een lagenmodel is niet volledig zonder een goede beschrijving van de te onderscheiden lagen, een specificatie van de inhoud van de lagen en van de communicatieregels tussen de lagen.

Ook mag de verantwoording, niet ontbreken. In de praktijk blijft het nogal eens bij alleen het eerste punt, dus alleen een beschrijving of plaatje van welke lagen onderscheiden moeten worden. Voorbeelden hiervan zijn: "Wij onderscheiden de volgende lagen: User Interface, Applicatie en Data" of "Wij passen Model View Controller (MVC) toe". Vergelijk ook de lagenmodellen in figuur 1 en 3, die op de namen na gelijk kunnen zijn, maar ook wezenlijk kunnen verschillen. Een team dat nauw samenwerkt en een gemeenschappelijk beeld heeft bij zo'n 'lagenmodel', kan daar nog een eind mee komen. Maar in de meeste situaties volstaat het niet. Niet voor de architect en niet voor de ontwikkelaar.

De architect die niet beschrijft wat de verantwoordelijkheid van een laag is en welke functionaliteit daar wel (en niet) onder valt, denkt daar mogelijk ook nauwelijks over na en laat dus allerlei vragen open. En hij loopt daarmee de kans dat de doelen, die hij met het lagenmodel wil bereiken, niet gehaald worden.

Ontwikkelaars, die niet meekrijgen welk type functionaliteit precies in welke laag terecht moet komen, zullen daar zelf keuzen in maken. Misschien met het gewenste resultaat, maar misschien ook niet. Zonder een volledige specificatie (lagen, inhoud/laag, regels, verantwoording) is het onwaarschijnlijk dat alle voordelen die met een lagenmodel behaald kunnen worden, ook echt gerealiseerd worden.
Door het ontbreken van standaards, kunnen we niet met alleen de namen van lagen volstaan. We zullen moeten uitleggen waar die laag voor staat, want de terminologie met betrekking tot lagenmodellen is niet eenduidig en soms ronduit tegenstrijdig. Een goed voorbeeld hiervan is het veelgebruikte begrip 'application logic' of ´application layer´, zoals in het lagenmodel in figuur 1. Application logic wordt door verschillende auteurs fors verschillend uitgelegd. Bijvoorbeeld, Craig Larman [Larman05] definieert het als volgt: "handles presentation layer requests; workflow; session state; window/page transition; …". Terwijl Thomas Earl [Erl05] er de volgende betekenis aan geeft: "an automated implementation of business logic …". Zo krijgt ook het begrip 'Domain layer' in verschillende lagenmodellen verschillende betekenissen.



Figuur 3: Een logisch lagenmodel met drie lagen die erg lijkt op die in figuur 1, maar toch wezenlijk zou kunnen verschillen. Zoals waar de besturing van de taak zit of waar de status wordt bijgehouden.

Benodigd: Een referentiemodel

Architecten die een lagenmodel opstellen moeten de keuze maken welke lagen onderscheiden moeten worden. Daarbij moet de kernvraag beantwoord worden: Waar staat een laag voor; welke typen verantwoordelijkheid of functionaliteit bevat die? In de literatuur die in de praktijk veel gebruikt wordt is redelijk wat te vinden om die vraag te helpen beantwoorden. Maar een compleet en implementatieonafhankelijk overzicht ontbreekt en de verschillende auteurs gebruiken verschillende en soms tegenstrijdige termen. Wat ontbrak is een referentiemodel voor de typering van functionaliteit, dat aan de volgende eisen voldoet:

- Het moet de typen logica beschrijven die onderscheiden kunnen worden in Enterprise Information Systems met een eenduidige naam, die de inhoud goed weergeeft en een duidelijke definitie per type logica: de verantwoordelijkheid met de bijbehorende functionaliteit;
- Het moet zo volledig mogelijk zijn;
- Het moet implementatievrij zijn, dus bijvoorbeeld niet alleen geschikt voor object oriëntatie of service oriëntatie.

Deze constatering heeft geleid tot de ontwikkeling van het 'Logica In Lagen' (LIL) referentiemodel. Om tot dit referentiemodel te komen zijn de lagenmodellen van toonaangevende auteurs en leveranciers geanalyseerd, met elkaar vergeleken en soms gecombineerd. Daarbij zijn de namen zo zuiver mogelijk gekozen, om (nog meer) homoniemen te voorkomen.

'Logica In Lagen' referentiemodel

Het doel van het LIL-referentiemodel is: Het zuiver definiëren en ordenen van soorten logica die veel voorkomen in Enterprise Information Systems, zodat een lijst ontstaat die ter referentie gebruikt kan worden bij het opstellen of analyseren van lagenmodellen.
De soorten logica worden in de onderstaande tabellen gedefinieerd door de beschrijving van de verantwoordelijkheid (tabel 1) en de typen functionaliteit (tabel 2).
De ordening van de soorten logica is hiërarchisch, in de vorm van een lagenmodel (tabel 1), waarbij de bovenste laag aansluit bij de gebruiker of 'cliënt' en de onderste lagen bij de infrastructurele voorzieningen. Verder bieden de onderliggende lagen herbruikbare functionaliteit aan de bovenliggende lagen.

Het LIL-referentiemodel

Het LIL-referentiemodel lijkt op een te implementeren lagenmodel, maar zo moet het niet gebruikt worden. Het is een referentiemodel op basis waarvan nagedacht kan worden over de vraag "Welke typen logica kunnen samengevoegd worden in een laag, of moeten juist gescheiden en verdeeld worden over meerdere lagen?" Dit afhankelijk van de kwaliteitseisen die aan een specifiek systeem gesteld worden.

Logicalaag	Verantwoordelijkheid
Presentatie logica	Het opbouwen en onderhouden van de communicatie met de gebruiker.
	Zoals het opbouwen van de user interface, het tonen van informatie en het afvangen van events/besturingsprikkels van de gebruiker.
	Het verwerken van events valt grotendeels binnen onderliggende vormen van logica.
	Ook bekend als: UI layer, View, ...
Taakspecifieke logica	Het coördineren van de taakuitvoering en het afhandelen van taakspecifieke functionaliteit.
	Zoals het bijhouden van de status van het proces, het regelen van pageflow of workflow en het uitvoeren van taakspecifieke berekeningen.
	De definitie van een taak is hier 'een eenheid van werk, die als een geheel moet worden uitgevoerd om een betekenisvol product voor de klant of gebruiker op te leveren'.
	Ook bekend als: Application logic layer, Process layer, Workflow layer, ...
Domeingenerieke logica	Het leveren van generieke (niet taakspecifieke, dus potentieel herbruikbare) functionaliteit, die te maken heeft met het interessegebied van de business.
	Zoals het ophalen, bewerken (berekenen, controleren) en opslaan van gegevens of het nemen van beslissingen.
	Ook bekend als: Business layer, Model, Application logic layer, ...
Infrastructuurabstractie logica	Het vertalen van functionele (technologie onafhankelijke) vragen in technologie afhankelijke vragen aan de infrastructuur.
	Zoals met betrekking tot persistentie & object relational mapping, security, logging, deployment.
	Ook bekend als: Data layer, Application logic layer
Infrastuctuur logica	Het leveren van breed herbruikbare, niet businessspecifieke diensten, zoals data persistentie, security, logging, deployment, etcetera.
	Ook bekend als: Technical Services layer, Technical Infrastructure layer, ...

Tabel 1: De soorten logica, geordend als lagen, met hun verantwoordelijkheid.

Logicalaag	Type Functionaliteit	Omschrijving/Voorbeelden
Presentatie logica	User Interface opbouwen en tonen	• Tonen van informatie
		• Aanbieden van invoer- en besturingsmogelijkheden
	Events afvangen	Onder welke omstandigheden moet op welke events gereageerd worden?
	Events afhandelen	Hoe moet op een event gereageerd worden?:
		• Afhandeling binnen eigen laag
		• Delegatie naar onderliggende laag
Taakspecifieke logica	Besturen van de taak	Wat moet er wanneer gebeuren? (afhankelijk van status)
		• Processtappen: pageflow, workflow, orkestratie
		• Transformaties
	Processtatus bijhouden	Welke selecties zijn gemaakt?
		Welke data ingevoerd of gewijzigd?
		Formele status (incl. 'commit').
	Processpecifieke bewerkingen (transformaties) uitvoeren	Selecteren en ordenen van gegevens.
		Processpecifieke rekenregels.
		Processpecifieke controles.
		Transactie managen.
Domeingenerieke logica	Ophalen en opslaan van gegevens.	Selecteren en ordenen van gegevens.
		Opslaan van nieuwe of gewijzigde gegevens.
		Verwijderen van gegevens.
	Generieke bewerkingen uitvoeren	Entiteit/Objectstatus bijwerken.
		Generieke rekenregels.
		Generieke controles.
		Generieke besturingsregels.
Infrastructuurabstractie logica	Data persistentie abstraheren	Waar en hoe is de data opgeslagen?
		Communicatie met database, foutafhandeling, ...
	Security abstraheren, ...	
Infrastructuur logica	Data technisch beheren	Database functionaliteit
	Programma's beheren en deployen	Applicatieserver functionaliteit.
		• load balancing service
		• system notification service
	Security leveren	Identification, authentication, ...

Tabel 2: Voorbeelden van veel voorkomende Typen Functionaliteit per Logicalaag

Het LIL-referentiemodel kan goed gebruikt worden bij het opstellen van een logisch of fysiek lagenmodel voor een systeem. Maar ook bij het analyseren van (leverancier)specifieke architectuurmodellen.
Bij het opstellen van een logisch lagenmodel kan de architect makkelijker kiezen welke typen functionaliteit uit het referentiemodel in een logische laag van het onderhavige systeem terecht moeten komen.
Verder kan de architect de lagen in zijn model eenvoudiger definiëren, door de verantwoordelijkheid en typen functionaliteit te beschrijven, gebaseerd op het LIL-referentiemodel.
Bijvoorbeeld, wat niet zichtbaar is in het lagenmodel van figuur 1, is een duidelijke definiëring van de lagen User interface, Application en Data.
De definitie van die lagen kan bijvoorbeeld grof worden gebaseerd op de logica lagen:

- UI-layer = Presentatie logica + Taakspecifieke logica;
- Application layer = Domeingenerieke logica;
- Data layer = Infrastructuur abstractie logica + Infrastructuur logica.

Maar waarschijnlijk is een fijnere indeling nodig, waarbij bijvoorbeeld de taakspecifieke typen functionaliteit worden verdeeld over de lagen User interface en Application.

LIL op fysiek niveau

Bij het opstellen van een fysiek lagenmodel kan het LIL-referentiemodel gebruikt worden om per type functionaliteit te bepalen hoe en waar die technisch geïmplementeerd kan worden.
Ieder type logica is op talloze manieren te implementeren; er zijn veel technieken, talen en leveranciers. Vooral Taakspecifieke logica en Domeingenerieke logica kunnen op heel veel verschillende plaatsen worden geïmplementeerd. Zoals in de user interface, de database (constraints of stored procedures), procedures in service componenten, session beans, backing beans, domeinklassen of in rule engines. Implementatie op verschillende plekken kan er voor zorgen dat functionaliteit die logisch bij elkaar hoort, wordt verspreid en gedupliceerd. En dat functionaliteit die logisch herbruikbaar is, fysiek op een plaats wordt geïmplementeerd waar die niet herbruikbaar is.
Bij het analyseren van (leverancier)specifieke architectuurmodellen kan worden bekeken welke typen functionaliteit uit het LIL-referentiemodel waar terecht zijn gekomen in het specifieke model. En of het specifieke model compleet en goed gedefinieerd is.
Zo valt bijvoorbeeld bij de analyse van de architectuur uit figuur 2 op dat de Controller laag zeer 'dun' is en zich alleen richt op de besturing van de paginaovergangen. Het grootste deel van de Taakspecifieke logica uit het LIL-referentiemodel zal dus in de View laag of in de Model, c.q. de service laag terechtkomen. Technisch kan het beide en als er geen aanvullende regels worden afgesproken, zal het willekeurig gebeuren. Verder is er weinig overeenkomst tussen de Model laag en de Domeingenerieke logica laag uit het LIL-referentiemodel. Als hergebruik van generieke domeinlogica bereikt moet worden, dan moet het lagenmodel uitgebreid worden.

Het Lagenmodel in de literatuur

Veel gevestigde auteurs geven het lagenmodel een prominente plaats binnen de architectuur van een digitaal informatiesysteem. Het boek 'Pattern-Oriented Software Architecture' wordt veelal gerefereerd als de belangrijkste bron, omdat daarin het 'Layers Pattern' uitgebreid wordt beschreven.

Voor de verantwoording van het LIL-referentiemodel kunt u terecht op www.onderzoek.hu.nl/publicaties en zoek vervolgens op auteur L. Pruijt. Voorstellen ter verbetering, ervaringen of adviezen zijn welkom (leo.pruijt@hu.nl).

Links

- *Bass, Len; Clements, Paul; Kazman, Rick / Software architecture in practice / Addison Wesley 2003*
- *Buschmann, Frank et all / Pattern-Oriented Software Architecture: A System of Patterns / John Wiley, 1996*
- *Earl, Thomas / Service–Oriented Architecture, Concepts, Technology and Design / Pearson 2005*
- *Earl, Thomas / SOA Design Patterns / Prentice Hall 2009*
- *Evans, Eric / Domain Driven Design / Addison Wesley 2004*
- *Fowler, Martin / Patterns of Enterprise Application Architecture / Addison Wesley 2003*
- *Larman, Craig / Applying UML and Patterns / Pearson Education 2005*
- *Lommers, Joost; Pruijt, Leo / Productgerichte architectuur / Software Release Magazine februari 2003*
- *Microsoft Patterns & Practices (www.msdn.microsoft.com) / Zoek op: Three-layered services application*
- *Oracle Technology Network (www.oracle.com/technology) / Zoek op: Oracle ADF Architecture*
- *The Open Group / TOGAF version 9 / Van Haren Publishing 2009 / Of als gratis download via www.opengroup.org/architecture*
- *Zeist, Bob van; et al / Kwaliteit van softwareproducten / Kluwer Bedrijfsinformatie 1996*

Leo Pruijt is als hogeschooldocent verbonden aan het lectoraat 'Architectuur van Digitale Informatiesystemen' aan de Hogeschool Utrecht.