



UvA-DARE (Digital Academic Repository)

Logic programming for knowledge-intensive interactive applications

Wielemaker, J.

Publication date
2009

[Link to publication](#)

Citation for published version (APA):

Wielemaker, J. (2009). *Logic programming for knowledge-intensive interactive applications*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

Samenvatting

Logisch programmeren gaat over het gebruik van mathematische logica voor het programmeren van een computer. In haar pure zin betekent dit volledig ontkoppelen van de declaratieve beschrijving van de oplossing en de oplostaak, het procedurele aspect. Bij gebrek aan efficiënte oplosstrategieën voor expressieve logicas en de moeilijkheid om inherent procedurele aspecten te beschrijven heeft deze pure vorm slechts een beperkt toepassingsgebied.

Een belangrijke stap naar een praktisch bruikbare taal voor logisch programmeren was de uitvinding van de programmeertaal Prolog. Deze taal is gebaseerd op een vereenvoudigde logica (Horn clauses) en een eenvoudige bewijsstrategie (SLD resolution). Prolog programma's kunnen gelezen worden als een declaratieve beschrijving van de oplossing en tegelijkertijd als een procedurele strategie om deze oplossing af te leiden. Met andere woorden, Prolog is in staat zowel kennis als procedures te beschrijven. Met name de beschrijving van interactie heeft procedurele aspecten. Deze combinatie in één programmeertaal (Prolog) maakt deze taal veelbelovend voor het type applicaties dat in dit proefschrift behandeld wordt: kennisintensieve interactieve applicaties.

Er bestaan enkele tientallen implementaties van de taal Prolog. Vrijwel allemaal concentreren deze zich op Prolog als regelgebaseerd systeem en laten zij interactie over aan andere programmeeromgevingen. Dit impliceert dat voor het bouwen van interactieve applicaties een *hybride* omgeving gebruikt moet worden. De noodzakelijke brug tussen Prolog en de taal die voor de interactieve aspecten gebruikt wordt maakt programmeren, en met name prototyping van interactieve applicaties, ingewikkeld en kan daarnaast zorgen voor een significant verlies aan performance. Dit proefschrift gaat over de vraag hoe een op Prolog gebaseerde infrastructuur er moet uitzien om kennisintensieve interactieve applicaties in een zoveel mogelijk geïntegreerde omgeving te kunnen schrijven.

Om deze vraag te beantwoorden zijn een aantal applicaties gebouwd. Het primaire doel van deze applicaties was *niet* hun architectuur, maar bijvoorbeeld onderzoek naar middelen om kennistechnologen te ondersteunen in het modelleren van kennis of onderzoeken hoe verschillende bestanden over museum collecties met elkaar geïntegreerd en doorzoekbaar gemaakt kunnen worden. De architectuur van deze applicaties was een secundaire doelstelling in de projecten waarvoor ze zijn gebouwd, maar de zoektocht naar de bouwstenen van

een geschikte architectuur staat aan de basis van dit proefschrift. Tijdens de bouw van deze applicaties zijn naar behoefte herbruikbare uitbreidingen aan Prolog en zijn bibliotheken gemaakt. In opvolgende applicaties zijn deze uitbreidingen getest en daarna verfijnd of, indien nodig, vervangen. Stabiele uitbreidingen zijn beschreven in wetenschappelijke publicaties en gepubliceerd als Open Source. Terugkoppeling vanuit de Open Source gemeenschap levert suggesties op voor alternatieven of verdere verfijning. In deze studie besteden we in het bijzonder aandacht aan de volgende drie probleemgebieden:

- *Kennisrepresentatie*

Hoewel Prolog een declaratieve lezing kent, heeft de taal een aantal beperkingen. Veel Prolog programmas die een correcte declaratieve representatie van het probleem zijn termineren niet of niet binnen een redelijke tijd. Frame geörienteerde talen (b.v., KL-ONE) voorzien in een vorm van kennisrepresentatie die beter geschikt is voor interactieve inspectie en manipulatie dan op logica gebaseerde talen.

Begin deze eeuw is de semantische web gemeenschap gekomen met de kennisrepresentatietaal RDF. Deze taal heeft een zeer eenvoudig datamodel, bestaande uit triples van de vorm {onderwerp, predikaat, waarde}. Daarbovenop zijn een aantal meer expressieve talen gedefiniëerd: RDFS en diverse dialecten van OWL. Het eenvoudige relationele model van RDF past uitstekend op Prolog en vormt sindsdien de kern van onze kennisrepresentatie.

- *Grafische toepassingen*

Veel Prolog systemen hebben geen voorzieningen voor grafische toepassingen en gebruiken een brug naar een conventionele grafische omgeving. Wij zijn van mening dat dit de productiviteit te veel schaadt, terwijl gebruik van de declaratieve aspecten van Prolog die ook zinvol zijn voor het beschrijven van interfaces en gebruikerinteracties bemoeilijkt wordt. Grafische omgevingen zijn doorgaans gestructureerd als object-georiënteerde systemen. Daarom hebben wij een uniforme koppeling met een extern object-georiënteerde systeem gerealiseerd en beschreven onder welke voorwaarden dezelfde architectuur voor andere object-georiënteerde systemen gebruikt kan worden.

- *Web-applicaties*

In de afgelopen jaren heeft het web zich ontwikkeld van een web van documenten naar een web met interactieve applicaties. Er is onderzoek gaande hoe dit soort applicaties door toevoegen van kennis bruikbaar gemaakt kunnen worden. Bovengenoemde integratie van museum bestanden is een voorbeeld. Mede doordat de voornoemde taal RDF bij uitstek geschikt is voor het integreren van diverse zeer verschillende kennisbestanden is het gebruik van Prolog voor de kennisaspecten van dit onderzoek een voor de hand liggende keuze. Als we Prolog echter inbedden in een architectuur die gebaseerd is op een traditionele webserver en pagina generatietaal (b.v., Apache met PHP) gaan veel van de voordelen voor snelle interactieve ontwikkeling verloren. Er is behoefte aan een meer 'Prolog-gezinde' manier om web-applicaties te maken.

Het is onze uitdaging om Prolog zodanig uit te breiden en van dusdanige bibliotheken te voorzien dat de taal geschikt is om grote en innovatieve applicaties geheel in Prolog te bouwen. In deze opzet worden alleen welbegrepen primitieven en benodigde interfaces naar besturingssysteem, netwerk en grafische primitieven in andere talen geschreven en als bibliotheek voor Prolog beschikbaar gesteld.

Overzicht van dit proefschrift Deel I van die proefschrift beschrijft uitbreidingen aan Prolog en bibliotheken voor Prolog die het mogelijk maken kennisintensieve interactieve applicaties geheel in Prolog te schrijven. Deze componenten worden waar mogelijk vergeleken met de aanpak voor soortgelijke problemen in andere (doorgaans Prolog) omgevingen. Deel II vormt een formatieve evaluatie van deze uitbreidingen: de beschreven cases tonen aan dat de gemaakte programmeeromgeving in staat is dit soort applicaties te ondersteunen, maar tegelijkertijd hebben zowel de bouw als de evaluatie van deze systemen geleid tot de ontwikkeling of verfijning van de uitbreidingen beschreven in deel I.

Deel I Hoofdstuk 2 beschrijft Triple20, een tool om ontologieën te browsen en beperkt te wijzigen. In strikte zin is Triple20 geen bouwsteen voor Prolog en hoort het dus niet thuis in deel I van dit proefschrift. Triple20 kan gebruikt worden als een bibliotheek, maar de voornaamste reden om dit systeem eerst te behandelen is dat dit hoofdstuk een goed inzicht heeft in de rol van kennis opgeslagen in het primitieve RDF model en hoe deze kennis gebruikt kan worden als fundament voor de architectuur van een interactieve applicatie. Het hoofdstuk behandelt een uitbreiding van het *model-view-controller* referentiemodel voor interactieve applicaties dat nodig is om de grote afstand in representatie tussen de kennisopslag en de interface efficiënt te overbruggen. Concurrency (threads; meerdere draden van executie) is hier een van de noodzakelijke bouwstenen. Triple20 vloeit voort uit de 'MIA' tools, beschreven in hoofdstuk 9.

De rest van deel I beschrijft de gerealiseerde uitbreiden aan Prolog en Prolog bibliotheken om ondermeer Triple20 mogelijk te maken. Het opslaan en opvragen van RDF triples, planning van complexe queries op deze database, koppelen van grafische bibliotheken, concurrency en infrastructuur voor het bouwen van web applicaties.

Het RDF triple model past naadloos op het relationele model van Prolog. Een triple database om onderzoek te doen naar eerder genoemde integratie van museum collecties dient minstens 10 miljoen triples kunnen bevatten, gelijktijdig vanuit meerdere threads gelezen kunnen worden en efficiënt kunnen zoeken naar tekst waarin sleutelwoorden voorkomen. RDF predikaten zijn georganiseerd in een of meerdere hiërarchieën welke in acht genomen dienen te worden om te kunnen redeneren over geïntegreerde bestanden. Dit type *entailment reasoning* dient ondersteund te worden met minimale overhead. Bij samengestelde zoekopdrachten (conjuncties) is de volgorde waarin de individuele RDF patronen vergeleken worden met de database van groot belang (vgl., database 'join' optimalisatie). Dit optimalisatieprobleem is het onderwerp van hoofdstuk 4 en is gebaseerd op metrieken van de RDF database. Een deel van bovengenoemde eisen is te specifiek om door de standaard Prolog dynamische database met maximale efficiëntie te kunnen worden opgevangen. De uiteindelijke

implementatie van de RDF database is onderwerp van hoofdstuk 3.

Hoofdstuk 5 gaat in op het koppelen van Prolog aan grafische bibliotheken. Zulke bibliotheken maken vrijwel zonder uitzondering gebruik van het object geïntendeerde paradigma. Dit hoofdstuk behandelt eerst een compacte interface die het mogelijk maakt objecten vanuit Prolog te manipuleren. Binnen object geïntendeerde omgevingen wordt nieuwe functionaliteit vaak gecreëerd door middel van afgeleide klassen. Bovenstaande interface geeft ons daar geen toegang toe vanuit Prolog, hetgeen zou betekenen dat het ontwikkelen van een grafisch programma in twee talen moet gebeuren: Prolog en de object taal om nieuwe klassen te maken. Dit schaadt de gewenste transparante ontwikkeling vanuit Prolog. Dit kan voor vrijwel alle object systemen opgelost worden door de nieuwe klasse in Prolog te beschrijven, waarbij de klasse declaraties gebruikt worden om een klasse binnen het object systeem te creëren en *wrappers* te genereren die de implementatie aanroepen binnen Prolog. Dit hoofdstuk vervolgt met integratie van Prolog data en non-determinisme in het object systeem en eindigt met een kritische evaluatie van dit object systeem voor Prolog.

Hoofdstuk 6 is een zijspng naar concurrency binnen Prolog. Traditioneel is onderzoek naar concurrency binnen Prolog veelal gericht op hoe een enkel programma geschreven in Prolog door meerdere CPUs uitgevoerd kan worden. De declaratieve interpretatie van de taal geeft aanleiding te verwachten dat dit beter gaat dan met meer procedurele talen. Helaas heeft dit werk niet veel ontwikkelaars overtuigd. Onze eisen zijn meer bescheiden: achtergrond berekeningen in grafische omgevingen, schaalbaarheid van web-servers en benutten van recente *multi-core* hardware. Op deze basis is een eenvoudig model van samenwerkende Prolog machines bedacht dat inmiddels door twee andere Prolog implementaties is overgenomen en de basis is voor standaardisatie binnen ISO WG17. Hoofdstuk 6 beschrijft het model, de consequenties voor de implementatie van Prolog en twee performance evaluaties.

Hoofdstuk 7 geeft een overzicht van wat er binnen SWI-Prolog is gebeurd om het maken van web-applicaties mogelijk te maken. Dit hoofdstuk geeft een samenvatting van de voorgaande hoofdstukken over RDF en plaatst dit materiaal in het bredere kader van web applicaties. Speciale aandacht gaat uit naar het parseren en representeren van web documenten (HTML, XML) en het genereren van zulke documenten vanuit Prolog. Waar mogelijk wordt onze infrastructuur vergeleken met PiLLOW, een andere infrastructuur voor web-programmeren binnen de Prolog gemeenschap. Ook wordt aandacht besteed aan web-servers (HTTP), waarbij met name code organisatie en schaalbaarheidsaspecten aan de orde komen. Hoofdstuk 7 benadrukt een aantal features die niet aan Prolog mogen ontbreken voor dit type applicaties: concurrency, geen limieten aan atomen, garbage collection van atomen en ondersteuning van de internationale UNICODE tekenset.

Deel II In het tweede deel worden een drietal applicaties besproken die hebben bijgedragen aan de ontwikkeling en evaluatie van de infrastructuur beschreven in deel I.

De eerste applicatie (PIDoc, hoofdstuk 8) is tegelijkertijd ook een deel van de SWI-Prolog infrastructuur. PIDoc is een omgeving voor *literate programming* in Prolog die, dankzij de embedded web-server, een interactieve omgeving biedt om documentatie te raadplegen en te verbeteren. PIDoc is onderdeel van de SWI-Prolog ontwikkeltools die noodzakelijk zijn voor

de productiviteit van de programmeeromgeving. Het bouwen van veel van deze tools heeft ook bijgedragen aan het testen en verfijnen van de infrastructuur, men name voor grafische applicaties.

Hoofdstuk 9 beschrijft experimenten met en de software architectuur van een prototype tool om annotatie en zoeken van multimediale bestanden met behulp van achtergrond kennis te bestuderen. Behalve XPCE (hoofdstuk 5) voor de grafische aspecten gebruikt dit prototype niets van de in deel I beschreven infrastructuur. Dit was de eerste tool waar RDF gebruikt werd om kennis uit te wisselen en waar het RDF datamodel tot in het design van de tool was doorgevoerd. In dit tool werd RDF opgeslagen in de Prolog dynamische database. Opname van dit hoofdstuk dient drie doelen: beschrijving van de applicatiecontext waarin tools worden ontwikkeld, benadrukken dat tool ontwikkeling en—in dit geval—onderzoek naar de rol van kennis in annotaties nauw geïntegreerd zijn en tot slot het formuleren van eisen voor de volgende generatie infrastructuur en tools. Dit werk vormde de directe aanleiding tot de ontwikkeling van Triple20 en een schaalbare RDF database, alsmede een extra motivatie om concurrency te introduceren.

Hoofdstuk 10 sluit deel II van dit proefschrift af met een beschrijving van de semantische zoek en annotatie toolkit ClioPatria. In dit hoofdstuk wordt de architectuur van ClioPatria, waarin alle technologie uit deel I samen komt⁷ gemotiveerd vanuit een *use case*. De *use case* is het bouwen van een prototype applicatie om beschrijvingen van kunstwerken die musea hebben samen met beschikbare achtergrondinformatie zoals diverse databases van kunstenaars, kunsttermen en geografische informatie doorzoekbaar te maken. De grote diversiteit in dataformaten, dataschemas en terminologie die te vinden is in de diverse musea is een van de uitdagingen. Een andere belangrijke uitdaging is om geschikte methoden te vinden om deze data te doorzoeken. De structuur van de geïntegreerde data is zo divers dat gestructureerde vragen (vgl., SQL queries) vrijwel niet te formuleren zijn. Om deze data toch doorzoekbaar te maken hebben we een algoritme ontwikkeld dat de data exploreert op basis van semantische afstand en de resultaten groepeerd naar de semantische relatie tussen de zoekterm en het gevonden kunstwerk. ClioPatria biedt een moderne web-gebaseerde interactieve interface gebaseerd op AJAX technologie van Yahoo! (YUI). De server is een Prolog web-server die alle services verzorgt.

Discussie De assumptie van dit proefschrift is dat logisch programmeren, en specifiek Prolog, geschikt is voor het maken van kennisintensieve en interactieve software. De opkomst van RDF en daarop gebouwde kennisrepresentaties heeft de bruikbaarheid van Prolog geloofwaardiger gemaakt. Eigenschappen van RDF modellen, zoals schaal en gewenste inferenties, maken een implementatie in puur Prolog minder geschikt, maar implementatie als bibliotheek in een andere taal (C) kan dienen als een natuurlijke uitbreiding van Prolog. De opkomst van het web heeft grote invloed op de architectuur van applicaties en past, mits voorzien van de benodigde web-bibliotheken, uitstekend binnen het domein waarvoor Prolog geschikt is.

⁷De grafische interface wordt alleen gebruikt als onderdeel van de Prolog ontwikkelomgeving.

Naast verdere standaardisatie van Prolog is het mogelijk van nog groter belang dat de Prolog gemeenschap standaarden vaststelt voor het representeren en genereren van (web-)documenten in XML en HTML. Alleen met zulke standaarden wordt het mogelijk resources voor het maken van web applicaties binnen de gemeenschap te delen. RDF kan deze rol van nature op zich nemen voor de kenniskant omdat de manier waarop het RDF datamodel aan Prolog gekoppeld dient te worden niet veel keuzes laat.

Dit proefschrift is een direct gevolg van de werkwijze t.a.v. software ontwikkeling binnen de vakgroep SWI (nu HCS), waar software ontwikkeling voor onderzoek uitgevoerd wordt door ontwikkelaars met een eigen onderzoeksdoelstelling: software architectuur. Deze werkwijze heeft duidelijke voordelen: (1) het biedt de mogelijkheid om wetenschappelijk geïnteresseerde software ontwikkelaars te voorzien van uitdagingen op architectuur niveau in het kader van een concrete vraag naar een applicatie en (2) ontwikkeling van de applicatie die het oorspronkelijke onderzoeksdoel dient gebeurt vanuit een zeer directe betrokkenheid. Korte lijnen en een korte ontwikkel- en evaluatiecyclus is voor een grote klasse van softwareproducten een productieve werkwijze. Dit geldt zeker voor academische demonstrators.

Met SWI-Prolog hebben we een productieve omgeving gerealiseerd voor ons onderzoek. Daarnaast hebben we, dankzij de brede acceptatie van de Open Source gedachte en getuige de populariteit van SWI-Prolog, een bijdrage kunnen leveren aan het verspreiden van het gedachtegoed van logisch programmeren in een veel grotere kring van academische en commerciële gebruikers. Tegelijkertijd blijft deze gebruikersgemeenschap ons motiveren en van problemen en oplossingen voorzien.