



Generating Integrity Preserving Associations

Reinier van Oosten

dutch



dutch





Gipa

- Gipa is geen MDA.
- Gipa bereikt hetzelfde vanuit een andere doelstelling:

- **Het borgen op instance niveau van de integriteit van domein modellen**



Het probleem

- The toestand van objecten kan veranderen.
- Veranderingen kunnen gevolgen hebben voor de toestand van andere objecten.
- De toestand van een object kan daardoor in conflict komen met de toestand van andere objecten.
- Dit is een klassiek integriteitsprobleem, onder andere resulterend in dangling pointers en memory leakage.



Oplossing

- Integriteitsproblemen kunnen worden opgelost met zelf-regulerende of homeostatische algorithmes.
- Ontwikkeling van homeostatische algorithmes kost veel tijd en is fout gevoelig.
- Hiervoor kunnen op taal niveau patterns worden gedefinieerd.
- Op basis van die patterns kan code worden gegenereerd.
- Het gebruik van de patterns kan worden gespecificeerd in een stuurformaat dat sterk geïnspireerd is op UML.



Gipa is geen MDA

- MDA genereert vanuit een UML model een domein model op implementatie niveau.
- GIPA begint bij het specificeren van een domein model in een XML sturbestand.
- Dit kan zeer goed naar UML worden omgezet via XMI.
- Lang niet elke UML constructie is toelaatbaar in Gipa.
- Op basis van het sturbestand wordt code gegenereerd.



Overview

- Integriteit van domein modellen
- Homeostatic of zelf regulerende benadering
- Patterns voor integrity preserving associations
- Specificatie van associaties
- Code generatie based on patterns and specification
- Biology inspired object-oriented modeling environment (Biome)



Integrity van object structuren

- Klassiek informatica probleem, maar niet opgelost.
- Dangling pointers-dangling objects-memory leakage.
- Gipa oplossing: een set regels die worden bewaakt middels homeostatic methoden.



Integriteit in domein modellen

- 3 typen domein objecten
 - Data objecten (Integer, Timestamp, String ...)
 - Structuur objecten (Collecties)
 - Model objecten



3 Rules

- Data objecten mogen nooit gewijzigd worden.
- Structuur objecten moeten privé blijven.
- Links tussen model objecten moeten altijd bidirectioneel zijn.



Data objecten

- Integers en vergelijkbare primitieven zijn niet muteerbaar.
- Veel data objecten, zoals strings en timestamps zijn dat vaak wel.
- Ook al zijn zij muteerbaar, doe dat nooit.
- Data objecten “weten” niet bij welk object zij horen en kunnen dus ook niet waarschuwen voor verandering.



Structuur objecten

- Model objecten worden ondersteunt door structuur objecten.
- Een structuur object “weet” niet bij welk object hij hoort.
- Muteren van een structuur object moet dus volledig gecontroleerd worden door de “eigenaar”.
- Dus een structuur object mag nooit worden gecommuniceerd.
- Wel mag een copy worden gecommuniceerd.



Model objecten

- Model objecten realiseren associaties met elkaar door naar elkaar te verwijzen.
- Langs deze verwijzingen kunnen opdrachten worden uitgegeven.
- Dit kan leiden tot veranderingen in de toestand van een object.
- Deze moet in een zelf regulerend systeem gerelateerde objecten daarvan op de hoogte stellen.



Voor- en Nadelen

- Voordelen
 - Het domein model is altijd stabiel.
 - Er kunnen een groot aantal taal-niveau patterns worden gedefinieerd die aan deze eisen voldoen.
- Nadelen
 - Coderen van deze regels is een arbeidsintensief proces dat door zijn complexiteit en herhaalbaarheid vervelend en foutgevoelig is.



Homeostatic modelling

- Het kennisframework van biologisch onderzoek kent 5 fundamentele organisatie dimensies en 5 fundamentele krachten.
- Alleen homeostasis of zelf-regulering is slecht terug te vinden in object-geörienteerde theorie.
- In complexe systemen is zelf-regulering essentieel voor stabiliteit.



Homeostatic modelling

- Verandering van focus:
 - Van: Die verandering moet doorgevoerd worden
 - Naar: Die stabiele eindsituatie moet gerealiseerd worden
- Subtiele verandering in terminologie
- Substantiele verandering in technologie



Parent and children

- Parent moet alle children kennen.
- Naïeve oplossing:
 - Parent heeft een collectie met children.
 - Als een child wordt toegevoegd, dan wordt deze eenvoudig aan de collectie toegevoegd.
 - Als de parent wordt gevraagd naar de children, wordt geantwoord met de collectie.



Wat gaat hier fout?

- Een child kan voorkomen bij meerdere parents.
- Een ander object kan de lijst met children manipuleren.
- Triggers voor en na toevoeging vinden dan alleen plaats bij toevoegen via parent.



Homeostatic oplossing

- Reactie op opdracht tot bereiken van een nieuwe stabiele situatie
 - Test eerst of de stabiele situatie al bereikt is.
 - Verwijder met de nieuwe situatie conflicterende toestands elementen
 - Verander de situatie tot de gewenste situatie
 - Geef aan gerelateerde objecten door dat zij een nieuwe stabiele situatie moeten realiseren, in evenwicht met die van dit object.



AddChild voor Parent

- Test of de collectie bestaat en maak deze eventueel aan
- Test of nieuwe child al in collectie is opgenomen
- Indien niet, voeg toe aan collectie
- Waarschuw nieuwe child dat dit nu zijn parent is



SetParent voor Child

- Test of parent al de gewenste parent is.
- Test of er een andere parent is
 - Indien ja, verbreek relatie met oude parent (ook homeostatisch)
- Zet de nieuwe parent.
- Indien deze niet nil, meldt de parent dat deze child moet worden toegevoegd.



Conclusie

- Niet schokkend.
- Veelvuldig te zien in stabiele systemen
- Het is geen uitgangspunt, en dat zou het wel moeten zijn.
- Coderen is complexer
- Wordt veelvuldig herhaald.

- Terrein van de implementation level patterns



Oplossing

- Een framework voor pattern based code generatie.
- Een model wordt gespecificeerd in een stuur bestand.
- De specificatie bepaalt:
 - De namespace en package van het model.
 - Een collectie van model object classen.
 - Een collectie van associaties tussen object classen.
 - Een collectie van rollen voor elke associatie die moeten worden geïmplementeerd door tenminste één class.



Associatie patterns

- Een associatie specificatie bepaalt het type associatie.
- De associatie specificatie bestaat uit de specificatie van de rollen.
- De rolspecificatie bepaalt de uiteindelijke namen van variabelen en methoden.
- De rolspecificatie bepaalt ook welke classen de rollen implementeren .
- De bij het type behorende associatie generator, genereert de code voor de verschillende classen.



Associatie patterns (2)

- Er is een aantal voorgedefinieerde associatie patterns.
- Het is eenvoudig nieuwe patterns toe te voegen, dan wel af te leiden van bestaande patterns.
- Het gedrag van de patterns kan in een enkele implementatie getest worden. Volgende implementaties voldoen aan dezelfde tests.



Enkele standaard patterns

- ToOne (simpel attribuut, uitsluitend te gebruiken voor enkelvoudige data)
- ToMany (meervoudig attribuut, uitsluitend te gebruiken voor meervoudige data)
- OneToOne
- OneToMany
- ManyToMany
- DoubleLink
- Tree



Specificatie

- Er is een (XML)stuurformaat gedefinieerd (GXD)
 - Classes met data attributen
 - Associaties met associatie type
 - Hierbij behorende rollen
 - Parameters en argumenten
 - Classen die de rollen implementeren



Resultaat

- Een class diagram kan worden gespecificeerd in gxd met behulp van een xml editor als oxygen
- Dit genereert Smalltalk code die 10 keer langer is dan de diagram code.
- Deze code is al getest.



Resultaat

- Een class diagram kan worden gespecificeerd in gxd met behulp van een xml editor als oxygen
- Dit genereert Smalltalk code die ± 10 keer omvangrijker is dan de diagram code.
 - Deze code is al getest.
 - GXD specificatie is inzichtelijker
 - GXD specificatie is eenvoudiger consistent te veranderen



Ervaring

- Er is een aantal projecten mee gewerkt
- Weinig extra code nodig
- Voorbeeld project: 400 regels GXD -> 3000 regels Smalltalk
- Hierbij werden enkele nieuwe patterns ontwikkeld.
- Hierbij werd eenvoudige een uitbreiding gerealiseerd: Object relational mapping volgens UnitOfWork (Fowler) met gebruikmaking van Glorp



Conclusie

- Gipa is een eenvoudig uitbreidbare techniek
- Levert veel winst in tijd en betrouwbaarheid
- Levert een meer inzichtelijk en aanpasbaar model

- Is beschikbaar voor Smalltalk
- Sluit aan bij het denken in patterns
- Is ook voor andere talen te realiseren, overigens met wisselende inspanning.