KU LEUVEN

FACULTEIT INDUSTRIELE
INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS GENT

Real-Time systemen samenvatting

Professor: dr. ir. Annemie Vorstermans

Maxim DEWEIRDT

Lennart VAN DAMME
Gilles CALLEBAUT
Stiaan UYTTERSPROT

Academiejaar 2015 - 2016

Inhoudsopgave

Hoofdstuk 1

Inleiding

1.1 Geef de definitie van real-time systemen

Een real-time systeem is een informatie verwerkend systeem dat reageert, in een eindig en gespecificeerde periode, op extern gegenereerde inputprikkels uit de fysische wereld. De reactie bij inputprikkels hangt dus niet enkel af van het resultaat maar ook van de responstijd. Niet tijdig een antwoord geven is even erg als geen of een fout antwoord.

1.2 Wat betekent hard, soft, real en firm real-time?

Hard real-time: Hierbij is het noodzakelijk dat het systeem de deadline haalt. Indien we dit niet halen kan het systeem op een verkeerde manier verder lopen of falen. vb1. landingssysteem dat automatisch opengaat voordat het vliegtuig landt. vb2. bij een kerncentrale zal men de moderator (vaak grafietstaven) automatisch in het reactorvat laten zaken indien er een ongecontroleerde kettingreactie van neutronen is.

Soft real-time: Dit zijn systemen waarbij de deadlines belangrijk zijn, maar af en toe 1 missen brengt de werking van het programma niet in gedrang. Er bestaat wel een mogelijkheid dat er kwaliteitsverlies is. Dit komt vaak voor bij systemen die het aantal behaalde deadlines willen maximaliseren of systemen waarbij we de vertraging van de taken willen minimaliseren. vb1. encoderen en decoderen van digitale tv-beelden. Vb2. Komt ook vaak voor bij data-acquisitie-systemen[1].

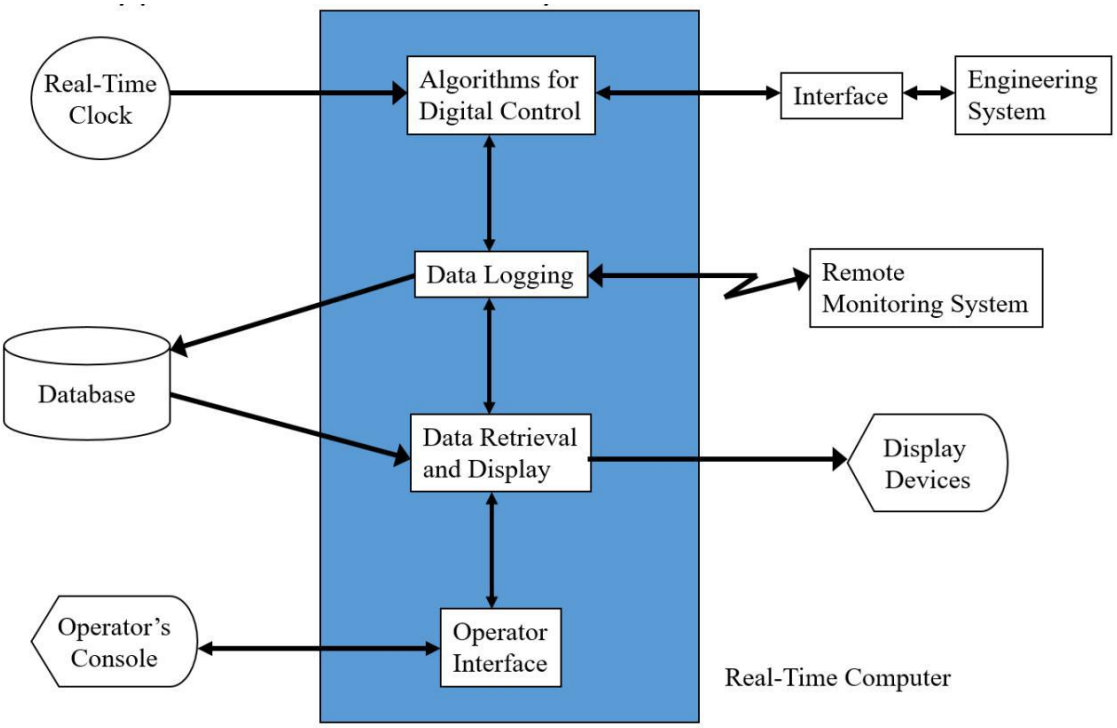
Real real-time systemen: Dit zijn systemen waarbij de deadline gehaald moet worden (= hard real-time) met zeer korte antwoordtijden. Vb. bij een raket gestuurd systeem moeten de deadlines gehaald worden (geen ruimte voor fouten) en willen we de raket zo snel mogelijk kunnen bijsturen bij externe storingen. Wat dus neerkomt op een hard realtime systeem met een zeer korte toegelaten response tijd.

Firm real-time systemen: dit zijn systemen waarbij we de deadline mogen missen (= soft real-time), maar zal het resultaat onbruikbaar worden. Firm real-time wordt vaak gecombineerd

[1] Data-acquisitie is het verzamelen van gegevens over parameters van bedrijsprocessen, waarbij hun gegevens direct online willen analyseren en presenteren.

met hard real-time gedeelten. Vb. een patient zijn beademingstoestel mag de periode van de ademhaling enkele seconden later tonen, maar niet meer dan dat.

1.3 Geef de figuur voor een typisch embedded systeem en bespreek.



Figuur 1.1: typisch embedded systeem

Real-time Clock : Bij real-time systemen moet het systeem vaak reageren op bepaalde evenementen in zijn omgeving. Om ze te monitoren, wordt er om de zoveel tijd een sample genomen door een meettoestel. Aan de hand van dit sample weet het RT-systeem hoe het moet reageren. Om het nemen van deze samples op een regelmatige tijdstip te laten verlopen, is er een real-time klok nodig.

Real-time computer : Dit is het blauwe gedeelte van de figuur. Dit bevat 4 modules : algorithms for digital control, data logging, data retrieval and display en het operator interface. De computer zorgt ook voor een sampling van de meettoestellen op een regelmatige basis(hiervoor heeft de computer de real-time klok nodig). De computer zal ook diegene zijn die controleert of alle systemen nog goed werken. Bij het falen van een systeem, kan de computer dit loggen/schrijven naar de database.

Operator's console : Dit zorgt ervoor dat er een manuele tussenkomst mogelijk is voor de gebruiker.

Operator interface : Via deze module kunnen het RT-systeem en de gebruiker interageren.

Database : Hier worden gegevens bijgehouden over de toestandsveranderingen van het RT-systeem. Indien men de reden achter een systeem faling wil of indien men meer info wil verzamelen over het RT-systeem voor administratieve doeleinden, kan men dit halen uit de database. Aan de hand van deze gegevens kan men zo bepaalde beslissingen maken voor het RT-systeem.

Algorithms for Digital Control : Het algoritme zorgt voor de besturing van het device.

Data logging : Deze module zorgt voor het opnemen van de toestandsveranderingen.

Data retrieval and display : Deze module gaat data uit de database en de data logger halen en deze aan de gebruiker tonen.

Interface : de interface zorgt voor interactie met het fysisch systeem.

1.4 Wat zijn de kenmerken van real-time systemen?

Groot en complex : Real-time systemen zijn vaak zo groot omdat deze systemen op een grote verscheidenheid aan real-world gebeurtenissen moeten reageren en ze zijn zo complex omdat ze continu wijzigbaar moeten zijn (noden en activiteiten kunnen veranderen). Een real-time systeem kan variëren tussen honderden tot miljoenen lijnen code in C, assembler of Ada(wordt vooral gebruikt in de ruimtevaart).

Manipulation of real numbers Er is een model nodig, die vaak kan worden opgelost via differentiaalvergelijkingen, rekening houden met sampling. Hiervoor is er een bepaalde nauwkeurigheid nodig (A/D-D/A-convertoren).

Uiterst betrouwbaar en veilig : Een real-time systeem kan bij falingen resulteren in schade, dood, ... We willen ook dat een real-time systeem lang meegaat zonder dat we al te veel updates moeten uitvoeren.

Gelijktijdige controle van aparte systeemcomponenten : Real-time systemen bestaan uit verschillende componenten die parallel kunnen opereren (vb. sensoren, actuatoren, ...). Soms zijn deze systemen op verschillende plaatsen aanwezig.

Faciliteiten om met special purpose hardware te interageren : We willen dat de devices op een algemene en betrouwbare manier kunnen aangesproken worden.

Een hoog taalniveau : Bevat een abstractie van implementatiedetails. Vaak hebben real-time systemen een antwoordtijd nodig binnen microseconde waardoor we deze hoge taalmogelijkheden niet zomaar mogen gebruiken. Men moet er ook voor zorgen dat de real-time software zo efficiënt mogelijk is geïmplementeerd.

Real-time faciliteiten : Bij real-time systemen is de responstijd cruciaal. Het is echter moeilijk om er altijd voor te zorgen dat de systemen, onder elke omstandigheid, op tijd een response geven. Om dit te realiseren, zorgt men ervoor dat de processor van real-time systeem voldoende capaciteit heeft om bij worst-case scenarioś geen vertragingen te veroorzaken. Voor een real-time applicatie te ontwikkelen, kunnen volgende zaken bekeken worden :

- Specificeren van tijdstippen: wanneer uitvoeren (sensor uitlezen), wanneer antwoord

- Opgeven van deadlines
- Reageren wanneer niet aan alle vereisten kunnen voldaan worden (deadline(s) gemist)
- Reageren op situaties waar tijdseisen dynamisch wijzigen (noodlanding vliegtuig)

Ondersteunen van numerieke berekeningen : Het real-time systeem moet ook ondersteuning kunnen geven bij numerieke berekeningen. Deze hebben ze nodig om input waarden te analyseren/controleren. Afhangend van het model, dat er wordt gebruikt, weet het real-time systeem wat te doen met deze waarden.

Hoofdstuk 2

Scheduling

2.1 Bespreek het simpel proces model (fixed-priority-scheduling, rate monotonic assignment, schedulability testen)

2.1.1 Simpel proces model

Het simple proces model heeft volgende karakteristieken:

- Vast aantal processen
- Periodisch met gekende periodes
- Processen zijn onafhankelijk van elkaar
- System-overhead, context-switching,... genegeerd (zero cost). Alle processen hebben een deadline gelijk aan hun periode (proces moet voltooid zijn vooraleer het volgende keer losgelaten wordt)
- Alle processen hebben een vast gekende worst case execution time

2.1.2 Fixed priority scheduling

Dit is één van de meest gebruikte scheduling technieken waarbij de prioriteiten van de processen pre-run-time (op voorhand) worden vastgelegd. De volgorde van de processen wordt at runtime bepaald door hun prioriteit.[1]

2.1.3 Rate monotonic assignment

Alle taken krijgen een (unieke) prioriteit gebaseerd op hun periode. Hoe korter de periode, hoe hoger de prioriteit. Als een proces gescheduled kan worden met fixed priority, dan kan het ook gescheduled worden met rate monotinic assignment. RMA wordt gezien als een optimale priority assignment.

[1] 1 is in deze cursus de laagste prioriteit

N	U(%)
1	100.0
2	82.8
3	78.0
4	75.7
5	74.3
10	71.8
∞	69.3

Tabel 2.1: Utilisatie waarden

2.1.4 Schedulability testen

Als we een set van processen hebben (met elk hun eigen prioriteit en periode), kunnen we via utilisation based analyse te weten komen of we deze processen al dan niet samen kunnen schedulen. Utilisation-based analyse (U) maakt gebruik van formule 2.1. Bij deze formule is N = aantal processen, C = computation-time(berekenings-tijd), T = periode.

$$U \equiv \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{\frac{1}{N}} - 1) \tag{2.1}$$

Om te weten of een set van processen schedulable is, moeten we onder volgende waarden zitten: Vb. voor 3 processen moeten we onder de 78% zitten als utilisation zie tabel 2.1.

Bij het testen van de schedulability, kan men ook rekening houden met de response time door gebruik te maken van response time analysis. Eerst moet men hiervoor de worst case response time R berekenen van taken via de formule 2.2 Vervolgens gaat men deze waarde vergelijken met de deadlines van de taak. Elke taak moet hier individueel bekeken worden.

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil .C_j \tag{2.2}$$

Met hp(i) verzameling taken met prioriteit hoger dan die van taak i.
Deze vergelijking moet opgelost worden via recurrente relatie:

$$w_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil .C_j \tag{2.3}$$

Hierbij is w_i^0 equivalent aan C_i. Deze formule wordt dan herhaald tot $W_i^n = W_i^{n+1}$. Dan is $R_i = W_i^n$. Als $R_i \leq T_i$ is dit proces schedulable. Zie sectie 2.2 voor een voorbeeld hierover.

2.2 Schedulability van proces sets kunnen bepalen (oef)

- MANIER 1: Utilisation-based analyse:

  Enkel voor taken waarbij D(Deadline) = T(periode). Het is een eenvoudige, voldoende maar niet nodige schedulability test.
  Om dit te berekenen maken we gebruik van de formule 2.1 op de pagina hiernaast.

Voorbeeld: *berekenen van schedulability via Utilisation-based analyse*

Proces	Periode T	ComputationTime C	Prioriteit P	Utilization U
a	50	12	1	0.24
b	40	10	2	0.25
c	30	10	3	0.33

Figuur 2.1: Info processen

We maken gebruik van de formule 2.1 op de linker pagina. Hieruit halen we dat:

$$U = \frac{12}{50} + \frac{10}{40} + \frac{10}{30} = 0.82 \tag{2.4}$$

Volgens de tabel 2.1 op de pagina hiernaast moeten we voor 3 processen onder de 0.78 zitten. De set voldoet hierdoor niet aan de test.

- MANIER 2 : via response time

Voorbeeld: *berekenen van schedulability via response time*

Proces	Periode T	ComputationTime C	Prioriteit P
a	7	3	3
b	12	3	2
c	20	5	1

Figuur 2.2: Info proces set D

Hier gaan we kijken of proces set D schedulable is. Hiervoor gaan we gebruik maken van de formules 2.2 en 2.3 op pagina 6. we beginnen met het proces met de hoogste prioriteit nl. proces A. Geen enkel proces heeft een prioriteit hoger dan A waaruit volgt:

$$R_a = C_a = 3 \tag{2.5}$$

R_a moet kleiner zijn dan of gelijk aan $P_a = T_a = 7$. Dit klopt. Dus we kunnen nu naar het proces gaan met de tweede hoogste prioriteit namelijk proces B. Hier is R als volgt berekend:

$$w_b^0 = 3$$
$$w_b^1 = 3 + \left\lceil \frac{3}{7} \right\rceil 3 = 6$$
$$w_b^2 = 3 + \left\lceil \frac{6}{7} \right\rceil 3 = 6$$

Omdat $W_b^1 = W_b^2$ mogen we stoppen en is R_b =6. $T_b = 12$.
$R_b \leq T_b$ klopt $\rightarrow$ proces b is ook schedulable.
Het laatste proces waarnaar we kijken is proces C. Hier wordt R als volgt berekend:

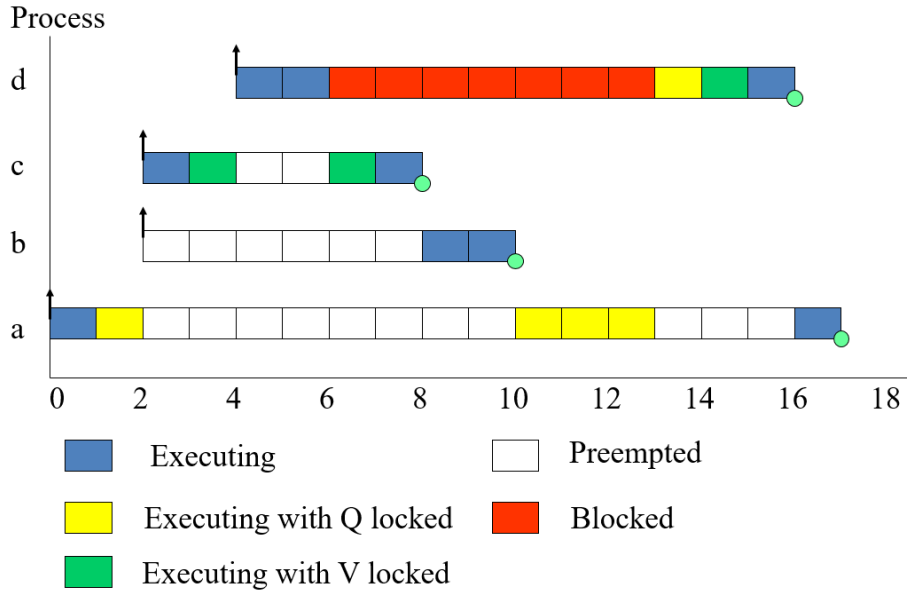
$$w_c^0 = 5$$
$$w_c^1 = 5 + \left\lceil \frac{5}{7} \right\rceil 3 + \left\lceil \frac{5}{12} \right\rceil 3 = 11$$
$$w_c^2 = 5 + \left\lceil \frac{11}{7} \right\rceil 3 + \left\lceil \frac{11}{12} \right\rceil 3 = 14$$
$$w_c^3 = 5 + \left\lceil \frac{14}{7} \right\rceil 3 + \left\lceil \frac{14}{12} \right\rceil 3 = 17$$
$$w_c^4 = 5 + \left\lceil \frac{17}{7} \right\rceil 3 + \left\lceil \frac{17}{12} \right\rceil 3 = 20$$
$$w_c^5 = 5 + \left\lceil \frac{20}{7} \right\rceil 3 + \left\lceil \frac{20}{12} \right\rceil 3 = 20$$

Omdat $W_c^4 = W_c^5$ mogen we stoppen en is R_c =20. $T_c = 20$.
$R_c \leq T_c$ klopt $\rightarrow$ proces c is ook schedulable. $\rightarrow$ proces set D is schedulable omdat alle drie de processen schedulable zijn.

2.3 Wat is het probleem bij scheduling als processen interageren? (+priority ceiling protocols)

2.3.1 Priority inversion

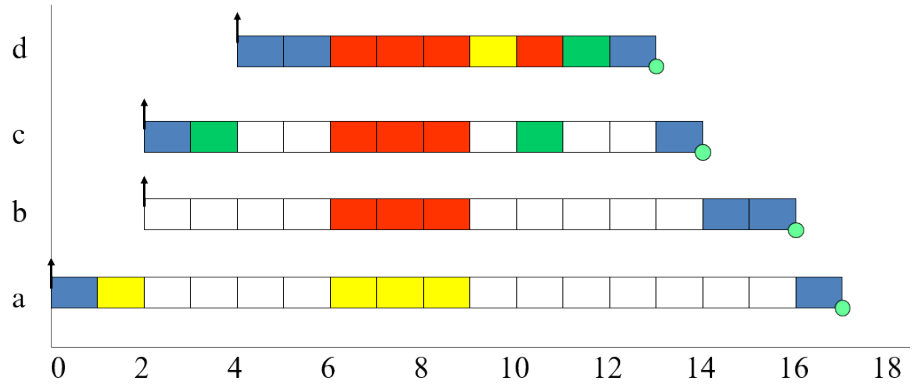
Bij het schedulen van processen kan het voorkomen dat een hoog prioriteitsproces moet wachten op een proces met een lagere prioriteit. Bvb. proces A met een lage prioriteit neemt een lock op resource Q. Vervolgens wordt proces D, die een hogere prioriteit heeft, gestart. Als proces D een lock wil nemen op resource Q, dan zal deze geblocked worden omdat proces A nog die resource vasthoudt. Proces D zal geblocked blijven tot proces A resource Q released. Indien er nog andere processen hiertussen worden gestart die een hogere prioriteit hebben dan proces A maar een lagere prioriteit dan proces D, dan zal (zoals op figuur 2.3) proces D geblocked worden door A,B en C. Om dit te voorkomen kan men gebruik maken van priority inheritance.



Figuur 2.3: Voorbeeld van priority inversion

2.3.2 Priority inheritance

Bij priority inheritance is de prioriteit van een proces niet meer vast. In het vorige voorbeeld zal proces 1, op het moment van de blocking van proces 2, de prioriteit krijgen van proces 2. Als proces 1 de resource A weer vrijgeeft, zal proces 1 weer zijn oude prioriteit krijgen.



Figuur 2.4: Voorbeeld van priority inheritance

2.3.3 Priority ceiling protocols

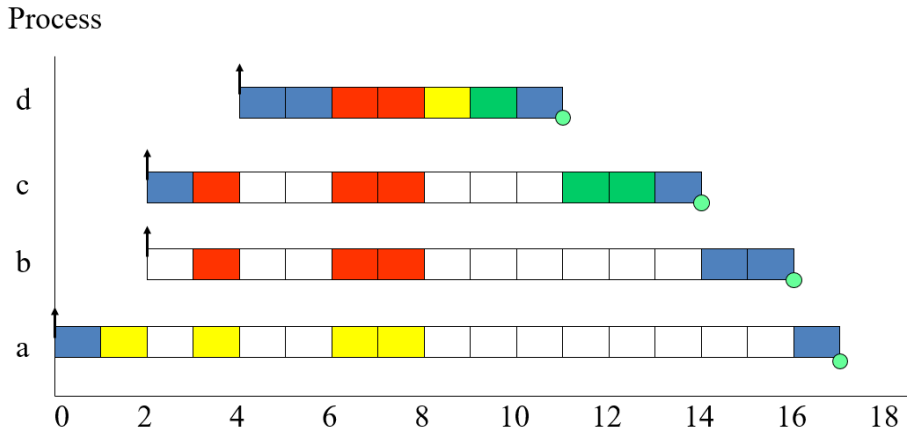
De priority ceiling protocols hebben 4 gemeenschappelijke eigenschappen (op een single processor) nl.:

- Een hoog-prioriteitsproces kan maximaal één keer geblocked worden tijdens zijn executie door een lager prioriteitsproces.
- Deadlock kan niet optreden
- Er kan geen transitieve blocking[2] plaatsvinden.
- Mutual exclusive toegang naar resources is gegarandeerd.

Original ceiling priority protocol : OCPP heeft volgende eigenschappen:

- Elk proces heeft een statische prioriteit toegekend gekregen
- Elke resource heeft een statische ceiling prioriteit gekregen die gelijk is aan de taak met de hoogste prioriteit die ooit deze resource zal consumeren.
- Elke taak krijgt een dynamische prioriteit die gelijk is aan zijn eigen statische prioriteit of aan het hogere prioriteitsproces dat de taak aan het blokken is (kijken wat het hoogst van de 2 is).
- Een taak kan enkel een lock nemen als zijn dynamische prioriteit hoger is dan de ceiling van alle gelockte resources.

[2] Transitieve blocking betekend dat een serie van blokkerende events plaatsvinden waarin lagere prioriteitstaken hogere prioriteitstaken blokkeren



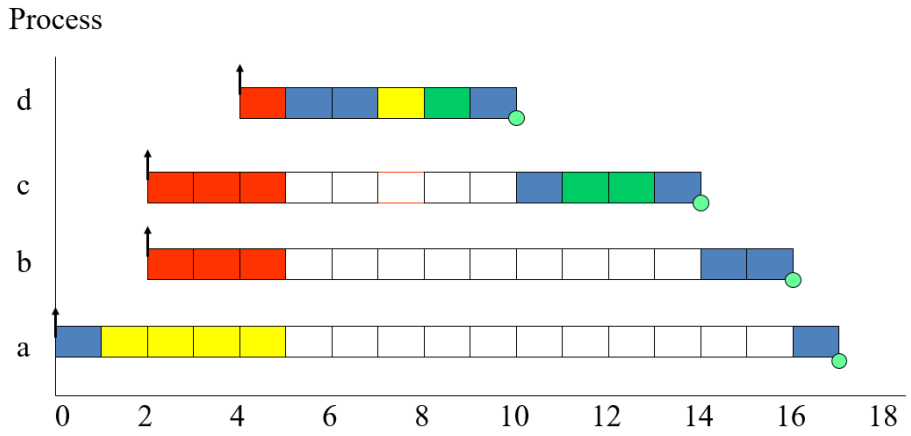
Figuur 2.5: OCPP example

Op figuur 2.5 op pagina 11 neemt resource A met de laagste prioriteit een lock op resource Q. Proces D is het hoogste prioriteitsproces dat ooit Q zal locken, waardoor de statische prioriteit van resource Q gelijk is aan de prioriteit van proces D. Proces B en C hebben een hogere prioriteit dan A, dus nemen nu processortijd in. Als proces D een lock wilt nemen op resource Q, kan hij dit niet doen omdat proces A een lock heeft genomen op resource Q. Omdat proces A proces D blockt, zal hij dezelfde prioriteit krijgen als proces D (zie derde eigenschap van OCPP). Een hoger prioriteits-proces wordt nu enkel geblocked gedurende de lock van een lager prioriteits-proces op de resource die beide processen willen locken.

We kunnen ook zien dat proces C direct wordt geblocked als hij resource V wilt locken. Dit komt door de 4de eigenschap van OCPP. De prioriteit van Q = prioriteit van D, maar de prioriteit van C is lager dan die van D waardoor C geblocked wordt.

Immediate ceiling priority protocol : ICPP heeft volgende eigenschappen:

- Elke taak heeft een statische(vaste) prioriteit
- Elke resource heeft een statische ceiling prioriteit gekregen die gelijk is aan de taak met de hoogste prioriteit die ooit deze resource zal consumeren.
- Elke taak krijgt een dynamische prioriteit die gelijk is aan de resource die hij op dat moment aan het locken is. Een hoog prioriteitsproces kan enkel geblocked worden in het begin door een lager prioriteitsproces.



Figuur 2.6: ICPP example

Resource Q zal ooit door process A en D gelocked worden. Proces D heeft de hoogste prioriteit waardoor resource Q dezelfde prioriteit krijgt als proces D. A zal als eerste een lock nemen op resource Q. Hierdoor krijgt A op dit moment dezelfde prioriteit als resource Q (en dus ook als proces D). A krijgt deze prioriteit zolang hij een lock neemt op Q.

2.4 Leg volgende formule uit:

$$R_i = B_i + C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \tag{2.6}$$

B: Worst-case blocking time for the process (if applicable)
C: Worst-case computation time of the process
D: Deadline of the process
I: The interference time of the process
J: Release jitter of the process
N: Number of processes in the system
P: Priority assigned to the process (if applicable)
R: Worst-case response time of the process
T: Minimum time between process releases (process period)
U: The utilization of each process (= $\frac{C}{T}$)
a-z The name of a process

- Gedurende R, zal elke hogere-prioriteitstaak j een aantal keer uitvoeren.
  $\lceil \frac{R_i}{T_j} \rceil$ = aantal releases van hoger prioriteitstaak j.
  $\lceil \frac{R_i}{T_j} \rceil . C_j$ = totale inteferentie door hoger prioriteitstaak j
  $I_j = \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil . C_j$ = totale interferentie op proces j van alle hogere prioriteitstaken.

$$R_i = C_i + I_i = C_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil . C_j \tag{2.7}$$

- Rekening houden met blocking door lager orde proces:

$$R_i = C_i + I_i + B_i = C_i + B_i + \sum_{j \in hp(i)} \lceil \frac{R_i}{T_j} \rceil . C_j \tag{2.8}$$

- Rekening houdend met release Jitter[3] per proces:

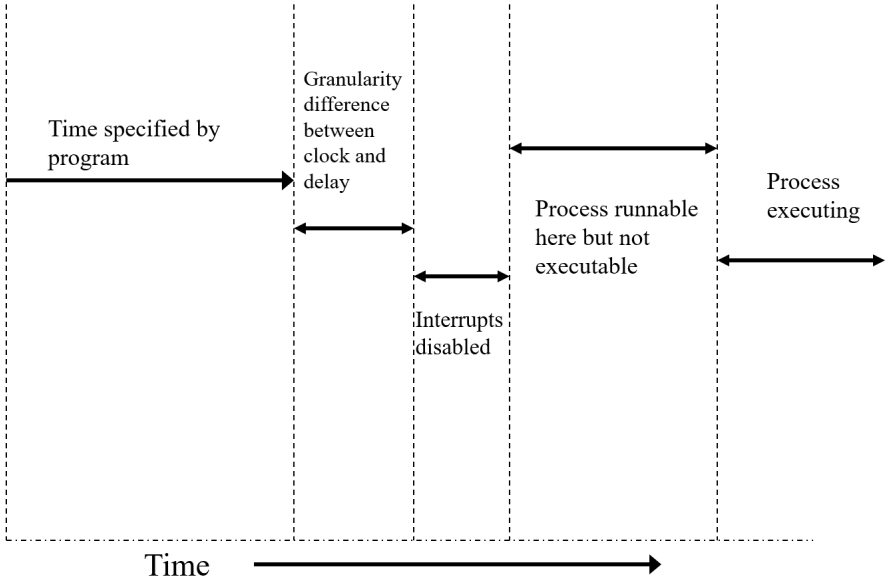
$$R_i = C_i + I_i + B_i = C_i + B_i + \sum_{j \in hp(i)} \lceil \frac{R_i + J_i}{T_j} \rceil . C_j \tag{2.9}$$

[3] Doordat er tussen de periodische taken ook sporadische taken zitten, is er een variatie in de taakuitvoering die we jitter noemen.

Hoofdstuk 3

Clock

3.1 Wat zijn de problemen bij het instellen van de delay voor een proces? Hoe los je dit op?

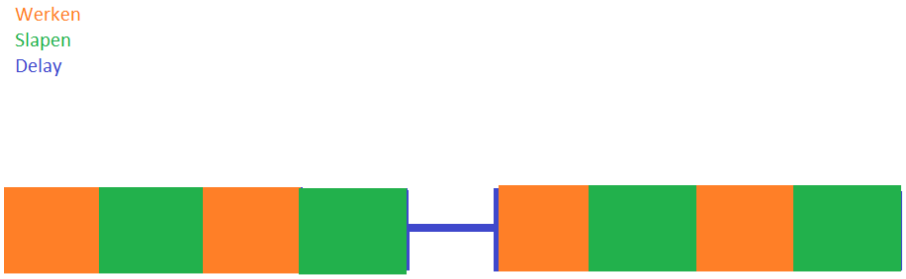


Figuur 3.1: Delays

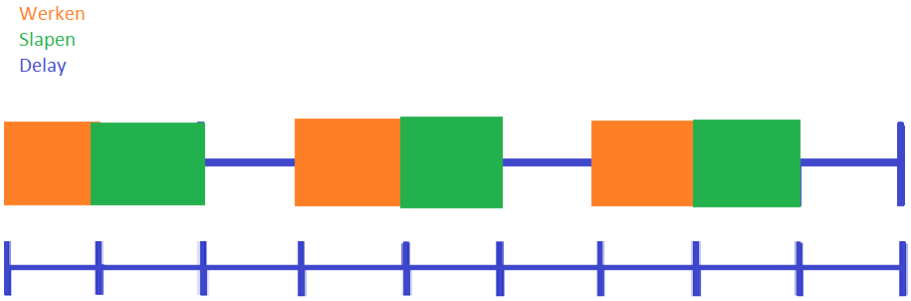
Real-time systemen moeten continue draaien en reageren op bepaalde gebeurtenissen in de omgeving. Om bepaalde gebeurtenissen te kunnen meten/op reageren, moeten enkele processen continue blijven draaien. Om te voorkomen dat deze processen via busy waiting blijven vragen of een bepaalde event is gebeurd, zal het proces om de zoveel tijd uitgevoerd worden. Deze tijd kan absoluut(op een vast tijdstip uitvoeren) of relatief(om de zoveel seconden) zijn. Via een klok weet het real-time systeem wanneer een proces moet uitgevoerd worden. Bij het opvragen van de tijd kan er al een tijdsverschil ontstaan, dit noemen we delays. Bij processen kunnen deze delays gaan opstapelen (dit noemen we cumulatieve drift). **Ideaal** zou zijn dat er geen delays zijn zoals in figuur 3.2 op de volgende pagina.

Figuur 3.2: ideaal

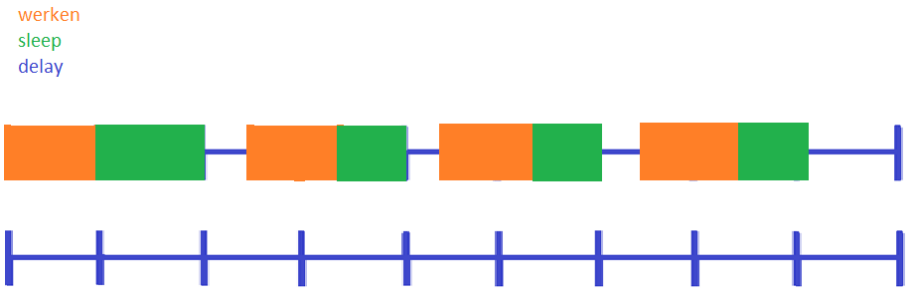
In realiteit is het praktisch onmogelijk om geen delays te krijgen. Op figuur 3.3 zien we een voorbeeld waarbij er een delay plaatsvind. Hier noemen we dit een **local drift**. Een local drift kunnen we niet vermijden. Als de local driften opstapelen, spreken we van **cummulatieve drift**. Op figuur 3.4 zien een voorbeeld van cummulatieve drift waarbij er na elke cyclus een delay blok bijkomt. De delays stappelen zich als het ware op. Cummulatieve drift is oplosbaar door de sleep en de delays op elkaar te laten afstemmen. Zo gaan we de sleep verminderen met de vorige delay zie figuur 3.5 op de pagina hiernaast.



Figuur 3.3: local drift



Figuur 3.4: Cummulatieve drift



Figuur 3.5: Cummulatieve drift oplossing

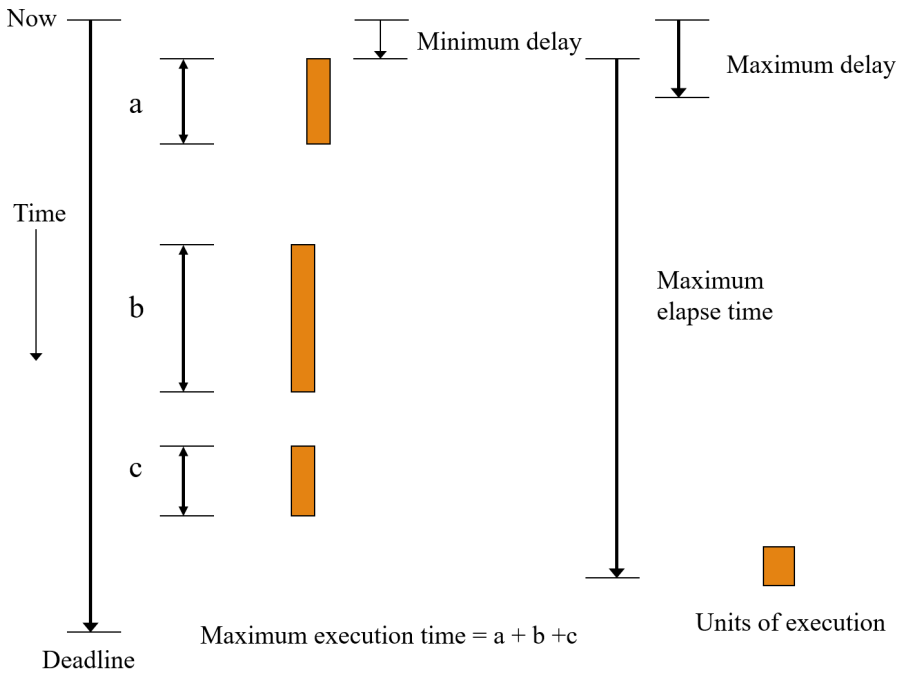
3.2 Hoe ontdek je en reageer je op het niet voorkomen van een extern event?

Door het proces, dat aan het draaien is, te laten wachten op het externe event. Indien dit niet binnen een opgegeven tijd plaatsvindt wordt het proces afgebroken (timeout). In Real Time Java gaat men op time-outs reageren via Timed of de klasse Timer.

Timed is een subklasse van de AsynchronouslyInterruptedException. Bij het verlopen van de timer wordt er een AsynchronouslyInterruptedException gegenereerd die dan opgevangen kan worden. Als we gebruik maken van Timed zal er een interrupt opgeroepen worden en geraken we in het AsynchronouslyInterruptedException blok. Daar kan de fout afgehandeld worden zoals de programmeur van het real-time systeem het wou.

Men kan in real-time java ook werken met de klasse Timer die een subklasse is van AsyncEvent. Als we gebruik maken van timer zal deze op de volgende taak een delay geven zodat de taak, die over zijn deadline zit, kan uitgevoerd worden.

3.3 Welke tijdseisen kan je stellen op processen?



Figuur 3.6: tijdseisen proces

- deadline: tijdstip tegen wanneer het ten laatste voltooid moet zijn.
- minimum delay: de minimum tijd voor het starten van het proces.
- maximum delay: de maximum tijd voor het starten van het proces.
- maximum execution time: maximale verstreken rekentijd.
- maximum elapse time: maximale totale verstreken tijd.

Deze tijdseisen kunnen onderverdeeld worden als :

- Periodisch : vast tijdsinterval tussen elke periode
- Sporadisch : er zit een minimum tijdsinterval tussen elke periode (minimum interarrival time). proces mag minder vaak opgestart worden, maar niet vaker (manier om beter te kunnen werken met aperiodische processen)
- Aperiodisch : helemaal geen regelmaat in tijd tussen elke periode. Dit is moeilijk uit te drukken in een scheduling.

3.4 Hoe detecteer je het overschrijden van tijdseisen en hoe ga je erop reageren?

- We kunnen bij het opstarten v/e proces een deadline recovery procedure mee laten opstarten. De recovery procedure wordt uitgevoerd met een delay van dat proces. Indien de deadline toch gehaald is, dan wordt de procedure nooit uitgevoerd. Nadeel hieraan is dat deze methode veronderstelt dat de taak mag gestopt worden indien de deadline niet gehaald is, om zo het recovery proces op te starten.
- Een andere manier om het overschrijden van tijdseisen te detecteren is door te werken met asynchronous events. (In Real Time Java[1] zal de virtuele machine een Asynchroon Event oproepen wanneer een periodische thread nog steeds aan het runnen is als zijn deadline verstreken is).
- De taak met een andere prioriteit laten uitvoeren bij het overschrijden van een tijdseis.

Sommige taken verwachten niet dat er gereageerd wordt bij het overschrijden van de deadline. Hier gaat het dan vooral om taken die een firm/soft real-time taken zijn. Indien we de deadline van een harde real-time taak overschrijden, kan dit tot gevolg hebben dat we de zaken doen die hierboven beschreven zijn. In het ergste geval laten we de taak stoppen.

[1] Opmerking: Real Time Java kent ook Sporadic Event Handlers maar die worden niet uitgevoerd bij het overschrijden van een deadline en wordt dus enkel voor soft deadlines gebruikt.

Hoofdstuk 4

Atomische acties, concurrent processen en betrouwbaarheid

4.1 Wat zijn atomische acties en wat zijn de eisen ervoor?

Bij atomische acties gaat men één of meerdere taken (= threads) uitvoeren zonder dat andere taken kunnen interfereren. Andere taken kunnen enkel zien wat de toestand van de taak (taken) was voor de atomische actie en na de atomische actie (alles wat er tussen gebeurd kunnen een andere taak niet zien). Een actie is atomisch als een groep van parallelle processen het volgende uitvoeren:

- Zich niet bewust zijn van het bestaan van andere actieve processen tijdens de actie en vice versa
- Niet communiceert met andere processen terwijl het een atomische actie aan het uitvoeren is
- Er geen toestandswijzigingen zijn, behalve die die het proces zelf veroorzaakt. Het proces zal ook zijn eigen toestandswijzigingen niet bekend maken tot de actie voltooid is (bvb: veranderen waarde van variabele)
- Als direct (zonder vertraging uitgevoerd) en ondeelbaar kan worden gezien door andere processen.

Atomische acties worden gebruikt voor :

- parallellisme
- soms moeten 2 processen 1 communicatie uitvoeren die niet mag onderbroken worden (bv. banktransactie)
- de betrokken processen moeten een consistente (niet-tegenstrijdig) systeem staat zien
- interferentie met andere processen vermijden

De eisen voor een atomische actie zijn :

- Goed gedefinieerde grenzen (begin, einde, scheiding tussen processen betrokken bij atomische actie en deze die dat niet zijn)
- Ondeelbaar
  - Mogen geen informatie uitwisselen tss interne en externe processen (behalve resourcemanagers, deze moeten namelijk de resource niet meer laten blokkeren als de atomische actie klaar is).
  - De processen mogen de atomische actie niet verlaten vooraleer alle deelnemende processen voltooid zijn (bij slagen van de atomische actie). Behalve bij recovery procedures.
  - Geen synchronisatie bij start: processen kan op verschillende momenten binnenkomen.
  - Waarde van gedeelde data na verschillende acties bepaald door strikte opeenvolging van acties in bepaalde volgorde
- mogen nesten zolang er geen overlap is met andere atomische acties
- Concurrency (gelijktijdig uitvoering van verschillende atomische acties moet mogelijk zijn).
- Er moet een mogelijkheid zijn om een herstelprocedure te programmeren.

4.2 Bespreek conversations, dialogs en colloquys.

4.2.1 Conversations

Als er zich een error voordoet in een atomische actie, dan zal de taak een rollback ondergaan(dit gebeurd via backward error recovery) om zo de atomische actie opnieuw te laten uitvoeren, maar met een ander algoritme (nieuwe start). Atomische acties zorgen ervoor dat er geen foute waarden worden meegegeven aan andere taken via communicatie uit de atomische actie. Als atomische acties op die manier werken, worden ze ook wel conversaties genoemd.

Hoe de conversaties werken:

- Bij het binnenkomen van het proces worden alle toestanden van het proces bewaard. De verzameling van alle toestanden van alle processen vormen de recovery line.
- Zolang we binnen een conversatie zijn, mogen de processen enkel communiceren met andere processen in de conversatie en met de resource managers.
- Om uit de conversatie te treden, moeten alle processen voor de acceptatie test geslaagd zijn
- Als alles is geslaagd bij de acceptatie test, worden alle recovery punten verwijderd en gaan alle processen uit de conversatie
- Indien de acceptatie test niet is geslaagd, worden alle processen in oorspronkelijke toestand hersteld en wordt er een alternatief algoritme uitgevoerd.
- Als alle alternatieven falen, recovery op een hoger niveau uitvoeren.

Indien de actieve processen er niet mee willen communiceren, is het niet noodzakelijk dat alle processen binnen zijn om de conversatie te vervolledigen. Als er toch communicatie gewild is: blokkeren en wachten tot komst proces of voortdoen. De voordelen van deze conversaties zijn :

- processen waarbij de deadline verstreken is, kunnen de conversatie verder zetten of verlaten.
- mogelijk om conversaties te specifieren zonder verplichte deelname

Conversations hebben ook nadelen :

- als conversatie faalt, dan ALLE processen herstellen en in alternatieve mode plaatsen.
- soms beter opnieuw proberen in plaats van over te gaan op alternatieve module

Conversations hebben ook het grote nadeel dat als een conversation faalt, alle processen terug hersteld worden en met een ander algoritme starten. Een proces kan niet zomaar uit de conversatie treden wat soms nodig is. Zo kan het bijvoorbeeld zijn dat een proces faalt als hij met een groep communiceert, hierdoor wilt het proces eens met een andere groep communiceren. Maar dit gaat niet zomaar met conversations. OPLOSSING: werken met dialogs en colloguys.

4.2.2 Dialogs

- Groep processen in atomische actie, bij fout wordt het recovery point hersteld en faalt de dialog (geen alternatieve modules)
- Dialog statements hebben drie functies:
  1. De atomic action identificeren
  2. Een globale acceptatie test declareren voor de atomic action
  3. De variabelen die bij de atomic action zullen gebruikt worden gaan specificeren

  .
- Elk proces dat wil deelnemen, definieert ook een DISCUSS-statement dat onderdeel is van de atomische actie en dat de actie een naam geeft: DISCUSS dialog name BY sequence of statements TO ARRANGE boolean expression; de reeks DISCUSS-statements vormt hier de recovery line
- RULE: een proces mag de dialog niet verlaten vooraleer alle actieve processen succesvol hun local en global acceptance test hebben afgelegd

4.2.3 Colluquys

- Bevat groep van dialogs
- Controleert de acties
- Beslist welke herstelactie wordt gestart bij een fout van een dialog
- Mogelijk om alternatief dialogs te definiëren met mogelijk een ander groep processen/andere acceptatietest

4.3 Bespreek atomische acties en forward error recovery.

Forward Error Recovery en Exception Handling wordt gebruikt voor real-time applicaties waar geen roll back mogelijk is. Als er een exception in een proces in een atomische actie optreedt, wordt de exception asynchroon opgegooid in alle processen die deelnemen aan de actie. Als er geen handler zit in één van de processen die bij de atomische actie wordt gebruikt of als een handler faalt, dan faalt de atomische actie met een standaard exception: Atomic Action Failure en wordt deze opgeroepen in alle betrokken processen.

Als er een exception wordt opgegooid, kunnen volgende voorzettingsmodellen gebruikt worden:

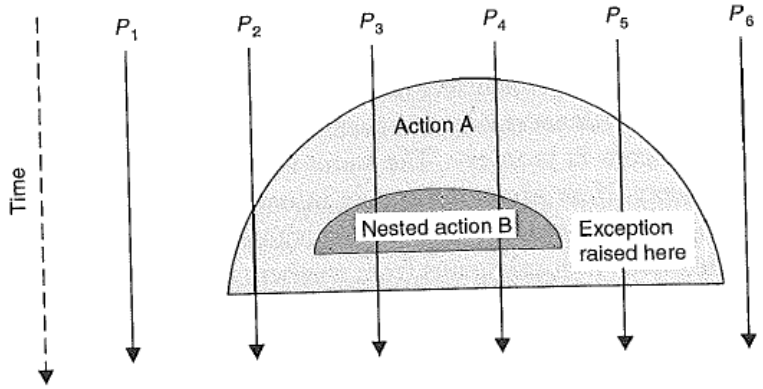
Voortdoen: Als een proces een exception veroorzaakt kan deze mogelijk door de handler worden opgelost waardoor het proces kan voortdoen alsof er niets is gebeurd.

Terminatie: Als een proces een exception veroorzaakt zorgt de handler ervoor dat het proces wordt beëindigd.

Hybride: De handler kiest afhankelijk van de situatie of het proces wordt voortgezet of beëindigd.

Het kan ook voorkomen dat meerdere processen tegelijk een verschillende fout opgooien. Indien men bvb. twee fouten heeft, met elk hun eigen handler, vormen die twee fouten samen een derde fout om zo aan te geven dat de fouten tegelijk zijn opgetreden. Om gelijktijdige fouten op te lossen, gaat men gebruik maken van exception trees. In deze tree gaat men de wortel zoeken van de kleinste subtree die alle opgetreden exceptions bevat. In het slechtste geval is dit de main root.

Het kan voorkomen dat een actieve taak in een geneste atomische actie een exception opgooit. vb. van een geneste actie kan u op zien in figuur 4.1.



Figuur 4.1: Nested atomic action

Als een exception wordt opgegooid, moeten alle taken meedoen aan de recovery actie. Maar de interne actie is echter ondeelbaar(nested action B). Om dit probleem op te lossen zijn er twee strategieën:

1. Wachten met de exception op te gooien tot de interne actie gedaan is. Maar indien de exception te maken had met een gemiste deadline kan dit geen goede strategie zijn. Het

kan ook voorkomen dat de exception wordt opgegooid omdat de interne actie in deadlock zat, hierdoor zullen we nooit de exception kunnen oproepen.

2. Laat de interne actie een voorgedefinieerde ‘Abortion Exception’ oproepen. De exception gaat dus eerst aan de interne actie melden dat er een fout heeft plaatsgevonden. Hierdoor weet de geneste atomische actie dat de alle acties gestopt zijn en dat de foutafhandeling kan beginnen.

Hoofdstuk 5

Design-processen

voor de eerste 2 vragen is het ook aangeraden eens hoofdstuk 15 in het boek te bekijken.

5.1 Beschrijf HRT-HOOD.

De meeste traditionele software development methoden maken gebruik van een life cycle model waarin volgende activiteiten worden gedefinieerd:

- Requirements specification: Een autoritaire specificatie van het vereiste functioneel en niet-functioneel gedrag wordt geproduceerd.
- Architectural design: Een top-level beschrijving van het voorgestelde systeem wordt ontwikkeld.
- Detailed design: Het ontwerp van het gehele systeem wordt gespecificeerd.
- Coding: implementatie
- Testing: De werkzaamheid van het systeem wordt getest

Bij hard real-time systemen heeft deze traditionele methode het grote nadeel dat timing problemen enkel herkend zullen worden tijdens het testen of zelfs pas na deployment. HRT-HOOD verschilt van deze traditionele design methoden. Het pakt immers direct de zorgen van hard real-time systemen aan.
HRT-HOOD staat voor Hard Real-Time Hierarchical Object Oriented Design. Hier ligt de focus bij de ontwikkeling van een logische en fysische architectuur gebruik makend van object-gebaseerde notatie.
Het is een architecturale design method. De architectuur wordt onderverdeeld in modules met goed gedefinieerde interfaces. Deze interfaces worden direct geïmplementeerd en de modules worden verder verdeeld.

Het **design proces** wordt aanzien als een opeenvolging van steeds specifiekere **commitments**, dit zijn eigenschappen van een systeemontwerp die ontwerpers op een meer gedetailleerd niveau NIET mogen wijzigen. **Obligations** zijn dan de aspecten van een design waar geen commitment wordt voor gemaakt. Lagere niveau's moeten deze behandelen.

In het vroege design kunnen er alreeds commitments voor de architecturale structuur van het systeem zijn in de vorm van object definities en relaties. Toch zal het gedetailleerd gedrag van de gedefinieerde objecten onderworpen worden aan de obligations.
Om een design te verfijnen moeten obligations in commitments omgezet worden. Deze verfijning wordt beïnvloed door beperkingen, vooral opgelegd door de uitvoeringsomgeving:

- resource constraints: processor speed, communicatie bandbreedte
- constraints of mechanism: interrupt priorities, task dispatching, data locking

Deze constraints staan vast voor zover de uitvoeringsomgeving niet wijzigbaar is. Obligations commitments en constraints hebben een belangrijke invloed op het architecturaal design. Daarom zal HRT-HOOD 2 activiteiten hebben:

- logical architecture design activiteit
  - Onafhankelijke van de constraints vanwege uitvoeringsomgeving.
  - Doel: vooral voldoen aan functionele eisen (de tijdseisen zijn vaak ook belangrijk voor de decompositie)
- Physical architecture design activiteit:
  - Vooral niet-functionele eisen
  - Vormt de basis om na te gaan of niet-functionele eisen zullen voldaan zijn na de implementatie
  - Kijken naar tijds- en afhankelijkheids-eisen, schedulability analysis: verzekeren dat het systeem, eens gebouwd, correct functioneert in zowel waarde als tijd.

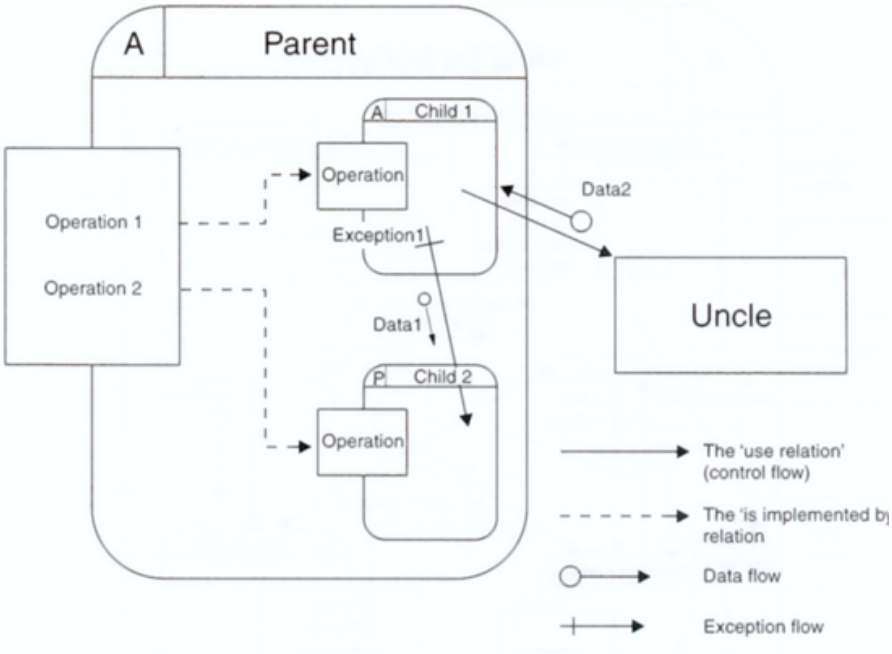
De physical architecture is een verfijning van de logical architecture, maar de ontwikkeling zal gewoonlijk een iteratief en samenlopend proces zijn waarin beide modellen worden ontwikkeld en aangepast. de analyse techniek gebruikt in de physical architecture wordt best zo vroeg mogelijk toegepast. Hiervoor kunnen initiële resource budgets gedefinieerd worden die later nog kunnen wijzigen wanneer de versie van de architectuur verder afgewerkt wordt. Op deze manier wordt een feasible design gevolgd vanaf het bepalen van de requirements tot de deployment.

HRT-HOOD vergemakkelijkt het logical architectural design van een systeem door verschillende object types aan te bieden:

- Passive: Passieve objecten zijn reëntrante objecten die geen controle hebben over wanneer hun operaties opgeroepen worden. Ze zijn niet in staat spontaan operaties van andere objecten op te roepen.
- Active: Actieve objecten hebben controle over de uitvoering van hun eigen operaties. Ze kunnen spontaan operaties van andere objecten invokeren. Ze zijn het meest algemeen en hebben geen restricties.
- Protected: Deze objecten hebben enkel controle over hun eigen operaties. Ze kunnen geen spontane invocatie van operaties bij andere objecten doen. Protected objecten mogen geen arbitraire synchronisatie-beperkingen hebben en moeten anlyseerbaar zijn voor de blokkeertijden die zij intstellen op hun oproepers.

- Cyclisch: cyclische objecten vertegenwoordigen periodische activiteiten. Ze zijn in staat spontane invokaties van operaties bij andere objecten te doen en hebben zeer beperkte interfaces.
- Sporadisch: Dit zijn objecten die sporadische activiteiten vertegenwoordigen. Ze kunnen spontaan operaties bij andere objecten invokeren. Elke sporadische heeft één operatie die opgeroepen wordt om de sporadic te invokeren.

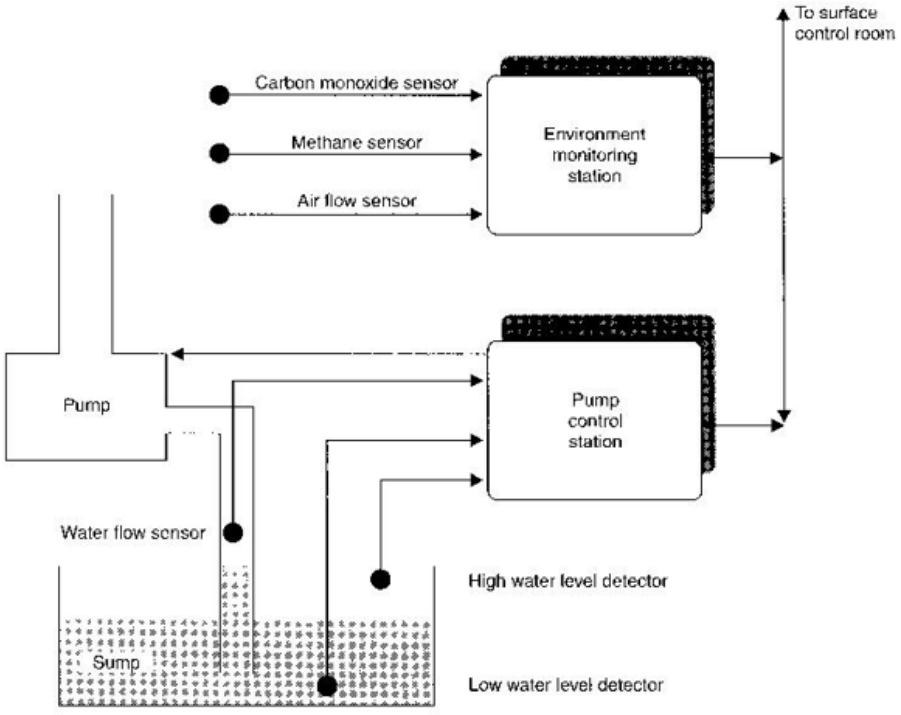
Op het terminale niveau (dit is na de full design decomposition) zal een hard real-time programma ontworpen met HRT-HOOD alleen cyclische, sporadische, protected en passieve objecten bevatten. Actieve objecten zijn niet toegelaten voor background activity omdat ze niet volledig geanalyseerd kunnen worden. Ze kunnen wel gebruikt worden tijdens decompositie van het main system maar moeten naar een ander type veranderd worden voordat het terminal niveau bereikt wordt.



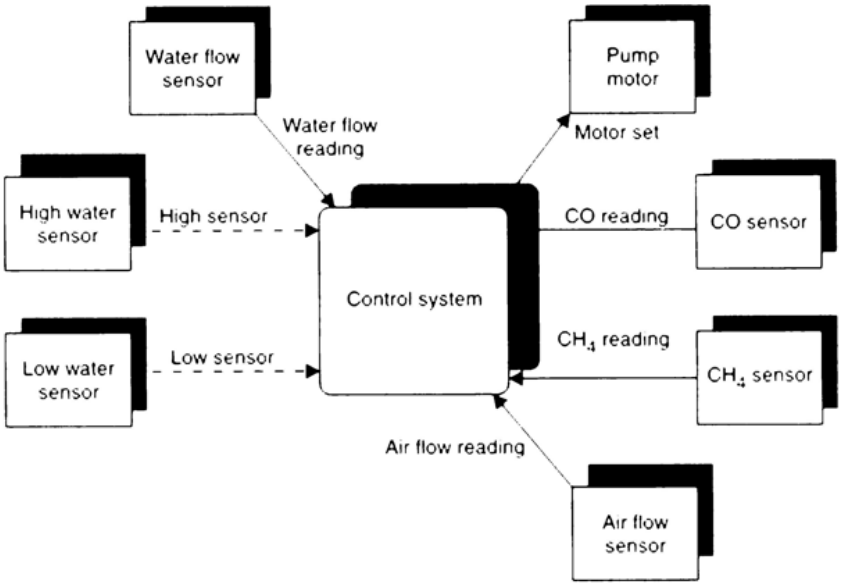
Figuur 5.1: HRT-HOOD diagrammatic notation

5.2 Illustreer HRT-HOOD aan de hand van het mijnvoorbeeld.

We willen met HRT-HOOD een mijnsysteem ontwerpen. Het systeem wordt gebruikt om water, dat zich beneden in de mijn bevindt, op te pompen naar het oppervlak. De pomp mag niet werken wanneer er een te grote hoeveelheid aan methaan gas in de mijn is om ontploffingen te vermijden. Een eenvoudig schematisch diagram van het systeem is gegeven in figuur 5.2 op de pagina hierna.
De relatie tussen het controlesysteem en externe apparaten wordt gegeven in figuur 5.3 op de volgende pagina. Hierbij communiceren alleen de hoog en laag watersensoren via interrupts (stippellijnen), alle andere devices worden ofwel gepolld ofwel direct bestuurd.



Figuur 5.2: Mine drainage control system



Figuur 5.3: Graph die external devices toont

5.2.1 Functionele eisen

- voor de pomp-operatie
  - pompcontroller monitort waterniveau in reservoir
  - als hoog niveau(of bij operatorrequest)
    - pomp aanzetten.
    - pompen.
    - als laag(of bij operatorrequest) pomp weer af.
  - stroming van water in pijpen detecteren als gewenst.
  - pomp enkel doen draaien als het methaangehalte beneden de kritische grens is.
- omgevingsmonitoring
  - detecteren methaangehalte (veiligheid tegen ontploffing).
  - detecteren CO gehalte (alarm als er teveel is).
- operator-interactie
  - systeem wordt bovengronds gecontroleerd via console van operator.
  - operator wordt verwittigd van alle kritische gebeurtenissen.
- systeem-monitoring
  - alle systeem-gebeurtenissen moeten opgeslagen worden in een archief-databank.
  - mogelijkheid opvragen en tonen op aanvraag.

5.2.2 Niet-functionele eisen

- timing
- betrouwbaarheid
- veiligheid

(de laatste 2 worden niet in het voorbeeld opgenomen)

Monitoring periods

De maximum periode voor het uitlezen van omgevingssensoren kan opgelegd zijn door de wet (hier voor alle sensors 100ms verondersteld). De CH4 en C-sensoren hebben 40ms nodig voor de uitlezing om beschikbaar te worden, de deadline is 60ms.
Het water flow object is periodisch en heeft twee rollen. Wanneer de pomp draait controleert het of het water stroomt en wanneer de pomp niet draait wordt gecontroleerd dat het water niet stroomt(controle pomp effectief gestopt). De periode is 1 seconde vanwege de traagheid van water en het resultaat van twee opeenvolgende uitlezingen wordt gebruikt om de toestand van de pomp te bepalen. Om te verzekeren dat de twee metingen effectief (ongeveer) een seconde van elkaar gescheiden zijn wordt er een deadline van 40 ms opgelegd (dus tussen de 960ms en 1040ms tijd

tussen twee metingen).
Waterniveau detectoren zijn event-driven en het systeem moet reageren binnen de 200ms. Er moet minstens 6 seconden tussen de interrupts van de 2 water level indicators zitten volgens de fysiek van de applicatie.

Shut-down deadlines

Om explosies te vermijden is er een deadline binnen dewelke de pomp moet uitgeschakeld worden bij het overschrijden van de kritieke threshold van het methaangehalte. Deze deadline heeft verband met de methaan-sampling periode, de snelheid waarmee methaan kan accumuleren en de veiligheidsmarge tussen het kritisch niveau en het explosieniveau. Via onmiddellijke uitlezing van de sensor kan de relatie uitgedrukt worden door de ongelijkheid $R.(T+D) < M$ waarin:

- R: accumulatierate van methaan
- T: sampling periode
- D: shut-down deadline
- M: veiligheidsmarge

Als "period displacement"gebruikt wordt is er een grotere tijdsperiode nodig: $R.(2T+D) < M$. Hoe langer de periode of deadline, hoe conservatiever de veiligheidsmarge.
We gaan er van uit dat een snelle stijging van het methaangehalte mogelijk is, daarom nemen we een deadline van 200ms (methaan hoog tot uitschakelen pomp). Dit kan als de methaan sensor rate 80 ms is met een deadline van 30 ms. Dit verzekert de correcte uitlezing van de sensor aangezien de displacement tussen twee uitlezingen minstens 50 ms is.

	Periodic/sporadic	'Period'	Deadline
CH_4 sensor	P	80	30
CO sensor	P	100	60
Air flow	P	100	100
Water flow	P	1000	40
High water level detector	S	6000	200
Low water level detector	S	6000	200

Figuur 5.4: Attributen van periodische en sporadische entities

Operator information deadline

Binnen de seconde moet de operator verwittigd worden als er een kritische hoeveelheid methaan of koolstofmonoxide is. Als de luchtstroom kritisch is moet binnen de 2 seconden geïnformeerd worden en als de pomp faalt binnen 3 seconden.

5.2.3 Logical architecture design

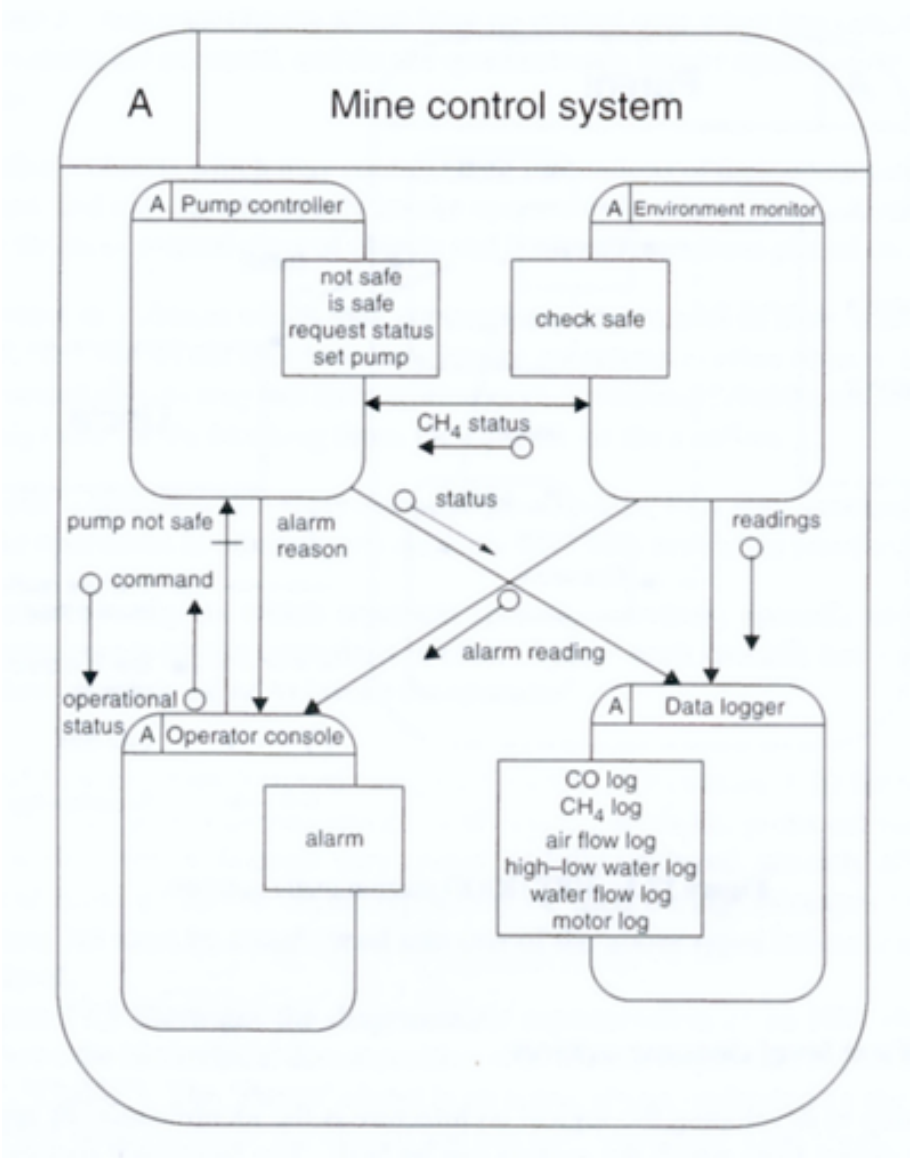
Hier worden de eisen in rekening gebracht die onafhankelijk zijn van de fysische constraints opgedrongen door de uitvoeringsomgeving.

First-level decomposition

De eerste stap om de logical architecture te ontwikkelen is de identificatie van de gepaste subsystemen van waaruit het systeem opgebouwd kan worden. Uit de functionele eisen kunnen we 4 gescheiden subsystemen halen:

- pomp controller
- environment monitor
- operator console
- data logger

Deze decompositie wordt weergegeven in figuur 5.5 op de pagina hierna. Elke component heeft een aantal voorziene en vereiste interfaces.



Figuur 5.5: First-level hierarchische decompositie van het controle systeem

- De pompcontroller kent 4 operaties:
  - 'not safe' en 'is safe': deze worden opgeroepen door de environment monitor
  - 'request status' en 'set pump': opgeroepen door de operator console

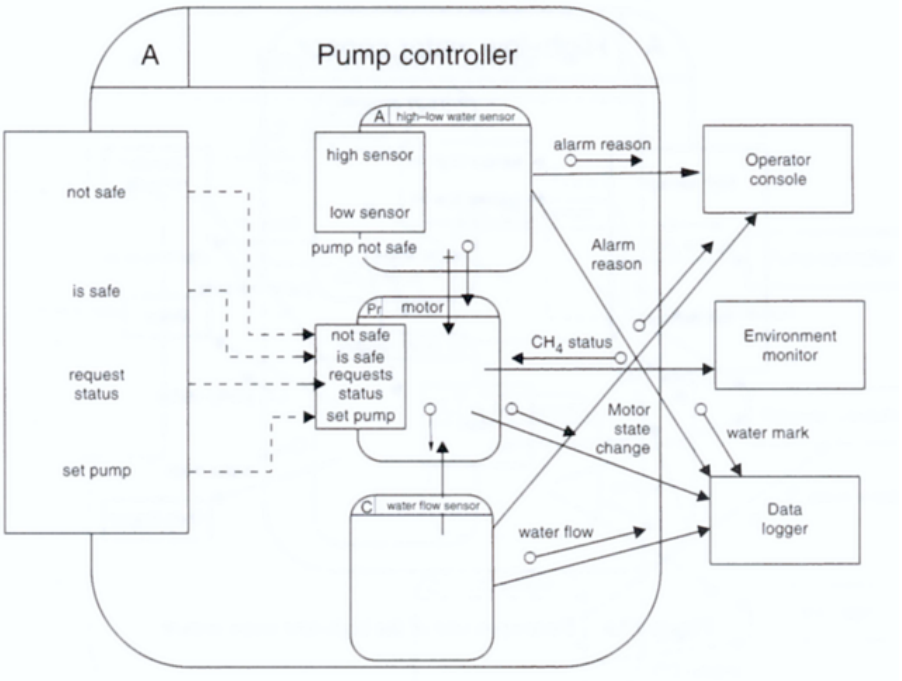
  Een bijkomend betrouwbaarheidskenmerk is dat de pomp controller steeds zal checken of het methaangehalte laag is voor het starten van de pomp(daarvoor wordt 'check safe' in de environment monitor opgeroepen).
  De pomp controller kan zien dat de pomp niet gestart kan worden of dat het water niet stroomt(terwijl de pomp aanstaat). Dan volgt er een operator alarm.

- environment monitor:
  - 'check safe'-operatie

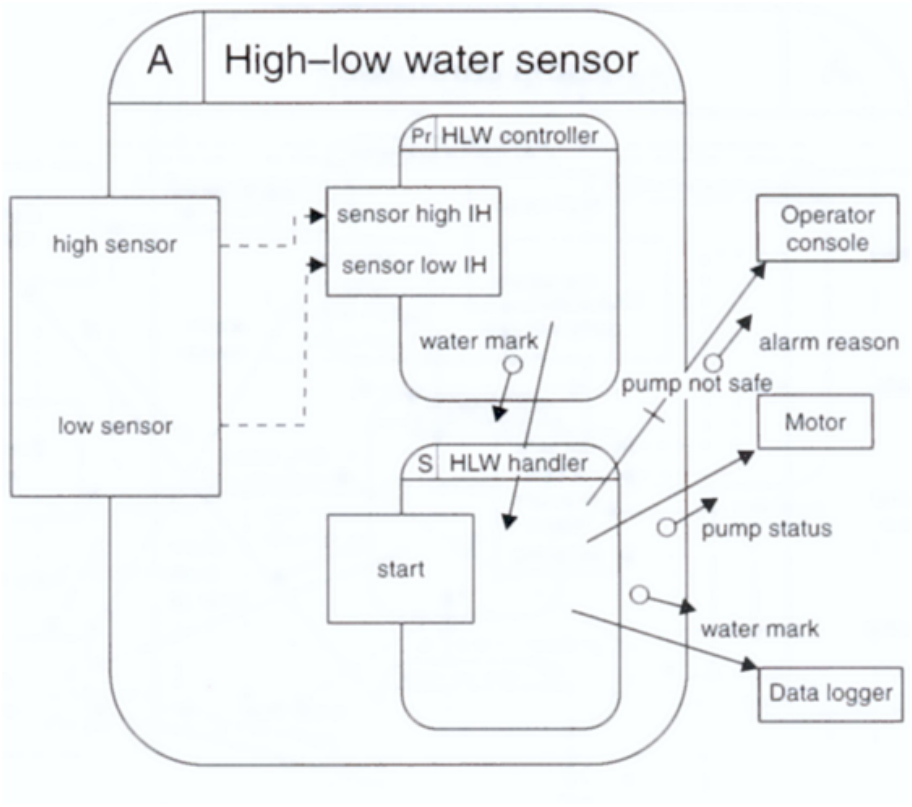
- operator console:
  - alarm operatie
  - opvragen toestand pomp
  - proberen watersensor te overriden door de pomp rechtstreeks te sturen (methaan-check blijft, exception als pomp niet aan kan).
- data logger: kent 6 operaties

Pomp controller

Figuur 5.6 toont het pomp object, dat gedecomposeerd wordt in 3 objecten(motor, water flow sensor, high-low water sensor). Aangezien dit object simpelweg reageert op commando's, mutuele exclusie een vereiste is voor zijn operaties en niet spontaan andere objecten oproept, is het een protected object. alle pompcontroller-operaties zijn geïmplementeerd door het motorobject. Het motorobject doet oproepen naar al zijn uncle-objecten.
het flow sensor object is een cyclisch object dat continue de flow van het water monitort en de high-low water sensor is een actief object dat de interrupts afhandelt die van de high en low water sensors afkomstig zijn. Het wordt gedecomposeerd in een protected en sporadisch object. (zie figuur 5.7 op de pagina hierna)



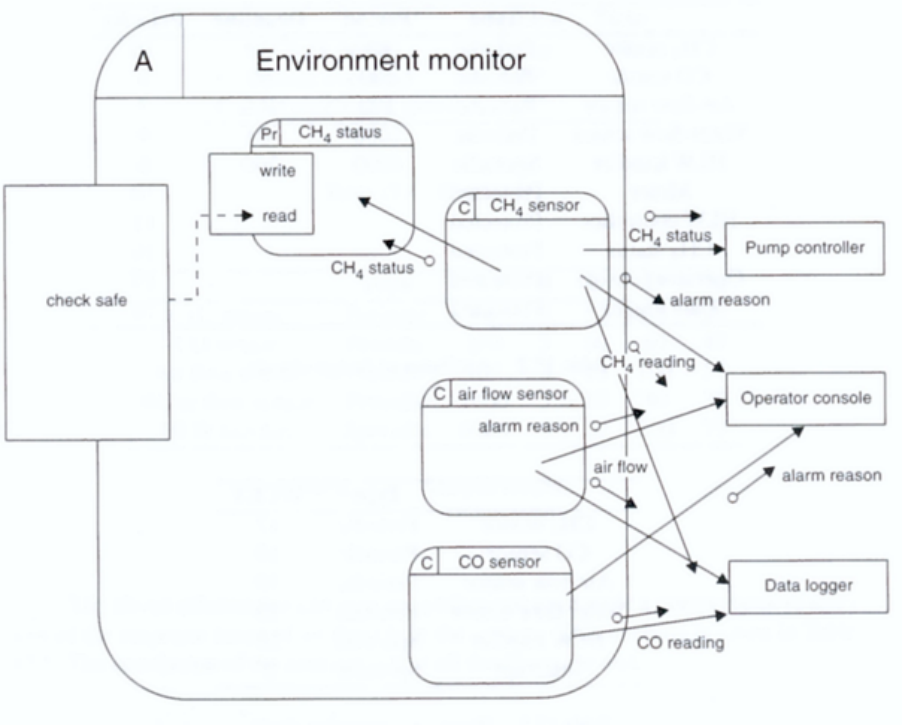
Figuur 5.6: hierarchische decompositie van het pomp object



Figuur 5.7: decompositie van de high-low water sensor

Environment monitor

De environment monitor is op te splitsen in 4 terminale objecten. Dit zijn 3 cyclische objecten voor monitoring CH4, CO en air flow. Het vierde is een protected object om toegang tot de concurrent gebruikte waarde van het CH4-gehalte te regelen.



Figuur 5.8: hierarchische decompositie van de environment monitor

Data logger en operator console

Wordt in deze case study geen rekening mee gehouden. Ze vertragen de real-time threads voor een beperkte tijd en er wordt verondersteld dat de interfaces enkel protected objecten bevatten.

5.2.4 physical architecture design

HRT-HOOD ondersteunt het ontwerp van physical architecture door:

- toe te laten tijdskenmerken te associëren met objecten
- een raamwerk aan te bieden waarbinnen de schedulability-aanpak kan gedefinieerd worden en de analyse van terminale objecten kan gebeuren
- levert abstracties waarbinnnen de ontwerper het afhandelen van tijdsfouten kan uitdrukken

Niet functionele eisen worden getransformeerd naar annotaties op methoden en threads.
voorbeelden kunnen zijn fixed priority scheduling, response-time analysis,...
Hier wordt fixed priority scheduling gebruikt. figuur 5.9 op de volgende pagina vat de timing attributen van de objecten geïntroduceerd in de logische architectuur samen.

	Type	'Period'	Deadline	Priority
CH_4 sensor	Periodic	80	30	10
CO sensor	Periodic	100	60	8
Air-flow sensor	Periodic	100	100	7
Water-flow sensor	Periodic	1000	40	9
High water handler	Sporadic	6000	200	6
Low water handler	Sporadic	6000	200	6
Motor	Protected			10
CH_4 status	Protected			10
Operator console	Protected			10
Data logger	Protected			10

Figuur 5.9: Attributen van design objects

Scheduling analyse

Bij de scheduling analyse wordt de code geanalyseerd voor worst-case execution times(meten of hardware modelleren). Aangezien er in het mijnsysteem geen zware berekeningen voorkomen kunnen we besluiten dat een processor met lage snelheid voldoende is.
in figuur 5.10 worden de representatieve waarden voor de worst case execution time gegeven in milliseconden.

	Type	WCET
CH_4 sensor	Periodic	12
CO sensor	Periodic	10
Air-flow sensor	Periodic	10
Water-flow sensor	Periodic	10
High water handler	Sporadic	20
Low water handler	Sporadic	20
Interrupt Low water	Sporadic	2
Interrupt High water	Sporadic	2

Figuur 5.10: worst-case execution time

De tijd die bij de analyse bekomen wordt voor elke thread bevat ook de tijd gespendeerd in andere objecten, gespendeerd in uitvoering van exception handlers en context switches.
Om het effect van de interrupt handlers te modelleren worden pseudo sporadische objecten geïntroduceerd (maximum handler execution time is 2ms).
Er zijn ook parameters opgelegd door de uitvoeringsomgeving, deze zijn te vinden in figuur 5.11 op de rechter pagina.

	Symbol	Time
Clock period	T_{CLK}	20
Clock overhead	CT^c	2
Cost of single task move	CT^s	1

Figuur 5.11: Overheads

De maximum blocking tijd voor alle taken komt voor wanneer de operator console een oproep doet op het motor object. We kunnen er van uitgaan dat de taak die de oproep doet van lage prioriteit is. We veronderstellen de worst case execution time voor deze protected operatie 3ms.
Nu kan al deze informatie samengevoegd worden om een tabel met de analyse van alle response tijden van alle taken in het systeem weer te geven. Deze analyse is te vinden in figuur 5.12. Hieruit kunnen we besluiten dat alle deadlines gehaald worden. De twee sporadische hebben dezelfde response times, aangezien tussen de interrupts die de taken releaset minimum 6 seconden zit. Dit betekent dat de sporadische taken nooit tegelijk uitgevoerd kunnen worden.

	Type	T	B	C	D	P	R
CH_4 sensor	Periodic	80	3	12	30	10	25
CO sensor	Periodic	100	3	10	60	8	47
Air-flow sensor	Periodic	100	3	10	100	7	57
Water-flow sensor	Periodic	1000	3	10	40	9	35
High water handler	Sporadic	6000	3	20	200	6	79
Low water handler	Sporadic	6000	3	20	200	6	79

Figuur 5.12: analyse resultaten

5.3 Beschrijf ROPES.

You use ropes to hang yourself after studying Real-time.

ROPES staat voor Rapid Object-Oriented Process for Embedded Systems. Ropes werkt op 3 tijdsschalen:

- macro: lengte project.
- micro: tijd nodig voor 1 increment of versie (4 tot 6 weken)
- nano: tijd nodig voor produceren, compileren, uitvoeren en testen van een heel klein gedeelte (30 minuten - 1 uur)

Voor het ontwikkelen van een ropes proces, kan men volgende vier fasen onderscheiden:

- Analysis
- Design

- Translation
- Testing

5.3.1 Analysis

De analysis fase bestaat uit het identificeren van de essentiële karakteristieken van alle mogelijke correcte oplossingen. De analysis fase kunnen we verder onderverdelen in:

- Requirements analysis : Hier worden alle vereisten van de klanten bekeken en worden deze in een gestructureerde en verstaanbare vorm gezet.
- Systems analysis : Hier worden er betere en accuratere modellen opgesteld dan vanuit het requirement analysis. De modellen worden hier gebasseerd op: de vereisten en hoe het systeem zich moet gedragen (dit zowel mechanisch als elektronisch als op software niveau).
- Object analysis : Dit is een totaal andere manier om het systeem te modeleren tijdens ontwikkeling. Object analysis bestaat uit twee subfasen nl. de structural object analysis en de behavioral object analysis. De structural object analysis identificeert de structuureenheden van de objecten en identificeert de organisatieonderdelen. Behavioral object analysis definieert de essentiële dynamische behavioral models voor de gekende klasses.

5.3.2 Design

De design fase voegt elementen toe aan de analysis fase zodat we een bepaalde oplossing kunnen definiëren en optimaliseren op basis van bepaalde criteria. De design fase kan in drie subfases onderverdeeld worden:

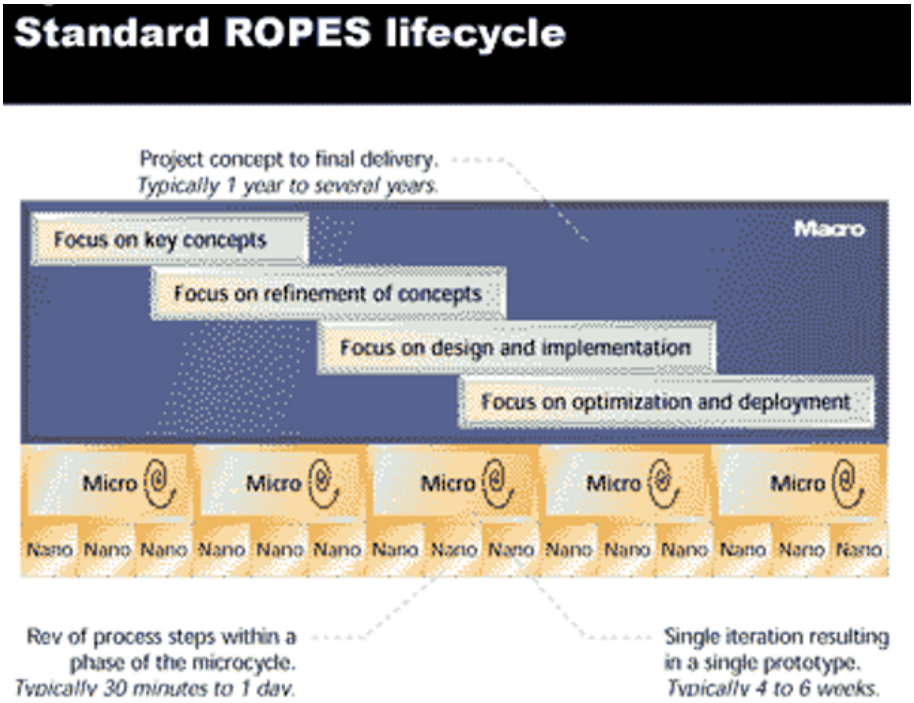
- Architectural design : deze subfase identificeert de grote, organisatorische stukken van het softwaresysteem. Het architectural design bestaat uit twee verschillende views nl. het deployment view en het concurrency view. Het deployment view organiseert de niet-uitvoerbare zaken (vb. source code) in verschillende secties waarop er dan individueel op gewerkt kan worden. Het concurrency view identificeert de gelijktijdige samenwerken tussen de objecten.
- Mechanistic design
- Detailed design : dit definieert de structuur en organiseert het interne van de individuele klassen.

5.3.3 Translation

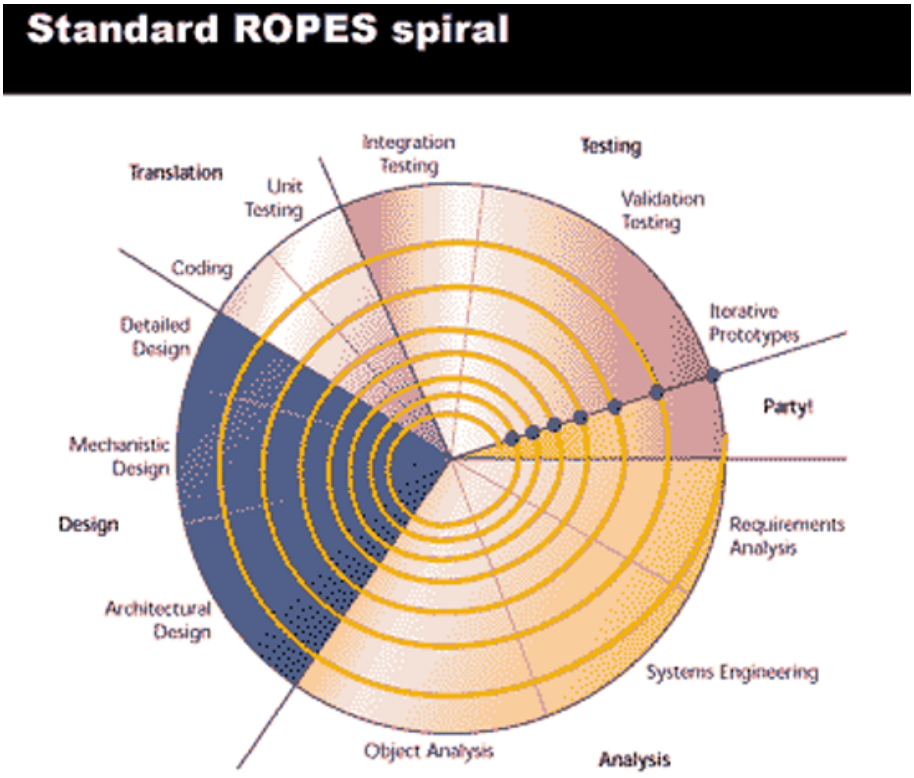
Deze fase zorgt voor een uitvoerbare en inzetbare realisatie van het design.

5.3.4 Testing

Hier gaan we kijken of hetgeen wat er in de translation fase uitkomt overeenstemt met wat er in de deisgn fase vermeld staat en wordt er gekeken of de implementatie alle criteria uit de analysis fase worden behaald.



Figuur 5.13: ROPES lifecycle



Figuur 5.14: Standard ROPES spiral

5.4 Beschrijf de Model-driven approach

5.4.1 Overview

Model-driven architecture (MDA) is a software design approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. Model-driven architecture is a kind of domain engineering, and supports model-driven engineering of software systems.

The model-driven architecture approach defines system functionality using a platform-independent model (PIM) using an appropriate domain-specific language (DSL).

Then, given a platform model corresponding to CORBA, .NET, the Web, etc., the PIM is translated to one or more platform-specific models (PSMs) that computers can run. This requires mappings and transformations and should be modeled too.

The PSM may use different DSLs or a general purpose language. Automated tools generally perform this translation.

Doel

Het is doel is een hogere abstractie brengen in de specificatie van een pogramma en meer automatisatie brengen naar ontwikkelen van platform-onafhankelijke programma's.

Aanpak

We kunnen dit verwezenlijken door modellen met verschillende niveau's van abstractie te ontwikkelen. Elk model moet een uitvoerbare modeltransformatie hebben om tot code generatie of model interpretatie te komen.

Volgens wikipedia:
OMG focuses Model-driven architecture on forward engineering, i.e. producing code from abstract, human-elaborated modelling diagrams (e.g. class diagrams)[citation needed]. OMG's ADTF (Analysis and Design Task Force) group leads this effort. With some humour, the group chose ADM (MDA backwards) to name the study of reverse engineering. ADM decodes to Architecture-Driven Modernization. The objective of ADM is to produce standards for model-based reverse engineering of legacy systems. Knowledge Discovery Metamodel (KDM) is the furthest along of these efforts, and describes information systems in terms of various assets (programs, specifications, data, test files, database schemas, etc.).

One of the main aims of the MDA is to separate design from architecture. As the concepts and technologies used to realize designs and the concepts and technologies used to realize architectures have changed at their own pace, decoupling them allows system developers to choose from the best and most fitting in both domains. The design addresses the functional (use case) requirements while architecture provides the infrastructure through which non-functional requirements like scalability, reliability and performance are realized. MDA envisages that the platform independent model (PIM), which represents a conceptual design realizing the functional requirements, will survive changes in realization technologies and software architectures.

Of particular importance to model-driven architecture is the notion of model transformation. A specific standard language for model transformation has been defined by OMG called QVT.

Model

Het model kan voorgesteld worden door een general purpose (UML) of domein specifieke taal (DSL). In het laatste geval spreken we dan ook over een domein specifiek model (DSM)

complexe systemen

We kunnen complexe systemen opbouwen door meerdere DSM's te beschrijven in meerdere DSL's, en dit tijdens de transformaties of voor de uitvoering. De kwaliteit van een MDE systeem kan worden getest a.d.h.v. model validatie, model checking en model-based testing.

Het verschilt verschil MDE en MDA (Model Driven Architecture) is dat MDA OMG's aanpak is van MDE. MDA zal zich vooral focussen op het oplossen naar meerdere platformen.

Modeltransformaties

M2M (model-to-model) gaat een bron model omzetten via transformatieregels naar een target model. OMG gebruikt hiervoor QVT wat een model transformation language specificatie is.

De transformatie kan dingen toevoegen:

- Bijkomende computational informatie (mensen), vb transformeren businees procesmodel: toevoegen info ivm regels, data, presentatie
- Technologische (platform) informatie (automatisch), vb datamodel naar Java-code

We moeten rekening houden met *change propagating transformation*:

- Non-destructive propagatie van wijzigingen van source model naar reeds bestaand target model
- enkel de wijzigingen (bv aanpassen databankstructuur)
- at run-time (zonder data te verliezen of integriteit te schenden) niet altijd mogelijk

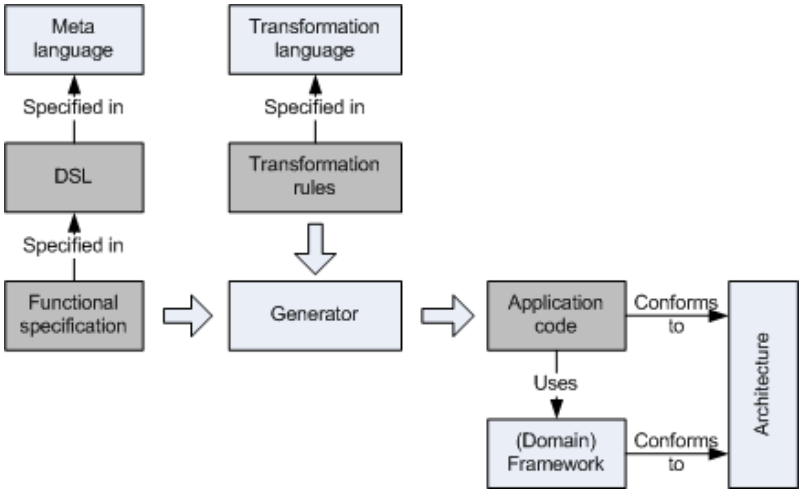
5.4.2 MDA

MDA is een aanpak voor het gebruik van modellen in software-ontwikkeling. Het beschrijft welke soort modellen te gebruiken en de relaties ertussen. Het basis-idee is het scheiden van de werking van een systeem van de details over hoe dat systeem de mogelijkheden van het onderliggende platform gebruikt.

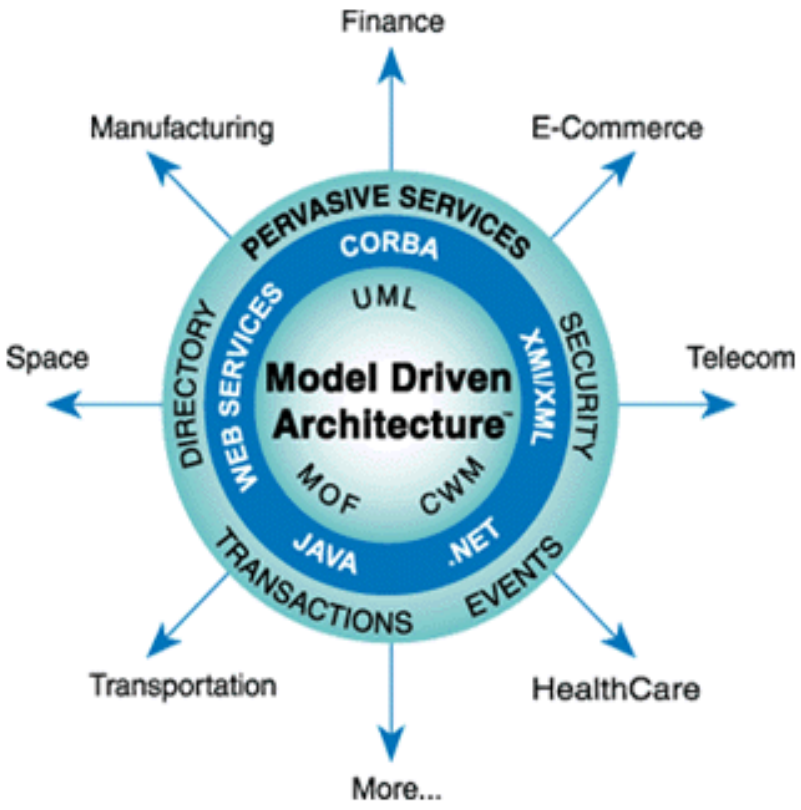
MDA lever:

- een aanpak waarbij systeemspecificatie onafhankelijk van ondersteunend platform
- aanpak voor specificatie van platformen
- aanpak voor transformeren van systeemspecificatie naar een specifiek platform

De belangrijkste doelstellingen zijn: portability, interoperability, reusability.



Figuur 5.15: MDE



Figuur 5.16: MDA

MDA inzichten

computation independent viewpoint:

- focus op omgeving van en eisen voor systeem
- details over structuur van systeem en verwerkingen binnen systeem : verborgen of nog niet

bepaald

- systeem wordt als black box beschouwd

platform independent viewpoint:

- focus op werking van systeem
- verbergen van details specifiek voor onderliggend platform
- toont alles dat gelijk blijft voor meerdere platformen
- general purpose modelleertaal zoals UML of taal specifiek voor domein

platform specific viewpoint:

- bijkomende details met betrekking tot een specifiek platform

MDA modellen

Drie default modellen corresponderend met zichten + platform-model

1. Computation independent model

- zicht op systeem vanuit computation independent zichtpunt
- toont geen details over structuur van systeem
- soms domein-model genoemd
- woordenschat gebruikt in specificatie van te bouwen systeem (domein)
- primaire gebruiker: mensen met domeinkennis, geen kennis nodig over verdere realisatie (die moet voldoen aan eisen gedefinieerd in CIM)
- CIM specifieert functie (extern gedrag) van systeem zonder constructie-informatie
- CIM: brug tussen domein-experten en IT-experten (ontwerp en ontwikkeling van systeem)

2. Platform independent model

- zicht op het systeem vanuit platform independent zichtspunt
- toont bepaalde platformonafhankelijkheid zodat bruikbaar voor meerdere verschillende platformen (wel van een gelijkaardig type)
- specificatie van systeem zonder implementatie-details
- systeem :
  - compositie: verzameling interne elementen
  - omgeving: verzameling element, disjoint met compositie
  - productie: elementen uit compositie produceren dingen (producten, services) die aan elementen uit omgeving geleverd worden
  - structuur: relaties tussen de verschillende elementen

3. Platform specific model

- zicht op system vanuit platform specific zichtpunt
- meer gedetailleerd dan PIM
- platform-specifieke elementen toegevoegd
- platform-model voor nodig

Relatie source target

Als het bestaande target-model wordt aangepast is het ook wenselijk dat het source-model wordt aangepast.
Het transformeren op zich bestaat uit een beschrijving van hoe elementen uit een source model moet vertaald worden naar elementen van het target model. Deze elementen bestaan uit LHS en RHS.

Markering Via een UML-klasse definieer je de stereotype Session in PIM, deze kan dan worden vertaald naar ene SessionBean en ondersteunende klassen in PSM.

Patterns Het strategie patroon in PIM wordt vertaald naar klassen in PSM.