

7. Dezvoltarea aplicatiilor software pentru sisteme embedded

7.1. Modele de dezvoltare software (V-Modell)

7.2. Reguli de dezvoltare software (ANSI, MISRA)

7.3. Standarde de dezvoltare software

7.1. Modele de dezvoltare software – V-Modell

7.1.1. Introducere in conceptul V-modell

Există numeroase abordări pentru dezvoltarea sistemelor, definite și proiectate care sunt folosite în procesul de dezvoltare sistemelor electronice. *Modelul V* (V Model) este un asemenea model, proiectat să faciliteze înțelegerea procesul complex asociat cu dezvoltarea sistemelor. In ingineria sistemelor este folosit pentru a defini un proces uniform pentru dezvoltarea produselor sau a proiectelor.

În general, un sistem de calitate livrat către utilizator, trebuie să:

- Îndeplinească sau să depășească cerințele utilizatorului/clientului
- Să se încadreze în limitele de cost prestabilite
- Să lucreze în mod eficient în infrastructura pentru care a fost proiectat

7.1. Modele de dezvoltare software – V-Model

7.1.2. Prezentare generala

Modelul V a apărut din necesitatea de a începe testarea încă din faza de proiectare a sistemelor. Acesta recomandă ca atât dezvoltarea sistemului cât și testarea acestuia să înceapă din același punct, referindu-se la aceleași informații. Acest fapt se datorează considerentelor economice, dat fiind faptul că cu cât o eroare în sistem este descoperită mai târziu, cu atât costul de corecție al acestuia crește. Estimările spun că un defect descoperit în timpul proiectării costă o unitate monetară pentru a fi corectat, același defect găsit chiar înainte de procesul de testare 6.5 unități monetare, în timpul testării 15 unități monetare, iar dacă defectul este descoperit după livrarea produsului, corectarea acestuia poate să ajungă să coste între 60 și 100 unități monetare (Figura 1)[3]. Prin urmare, cu cât se începe mai repede testarea sistemului, cu atât cresc șansele ca eventualele erori să fie descoperite din timp, fără să aibă un impact negativ asupra costului.

Conceptul modelului V a fost dezvoltat simultan, dar independent în Germania și în Statele Unite ale Americii la sfârșitul anilor '80 [2]. Modelul german a fost dezvoltat de IABG în Ottobrunn, în colaborare cu Oficiul Federal de Tehnologie de Apărare pentru Ministerul Apărării. În vara anului 1992 proiectul a fost preluat de Ministrul Federal de Interne [1]. Modelul SUA a fost prima dată documentat în 1991 la Consiliul Național al Ingineriei Sistemelor, și a fost dezvoltat pentru sisteme de sateliți, care implicau sisteme hardware, software și interacțiune cu un utilizator uman [2].

7.1. Modele de dezvoltare software – V-Model

7.1.2. Prezentare generala

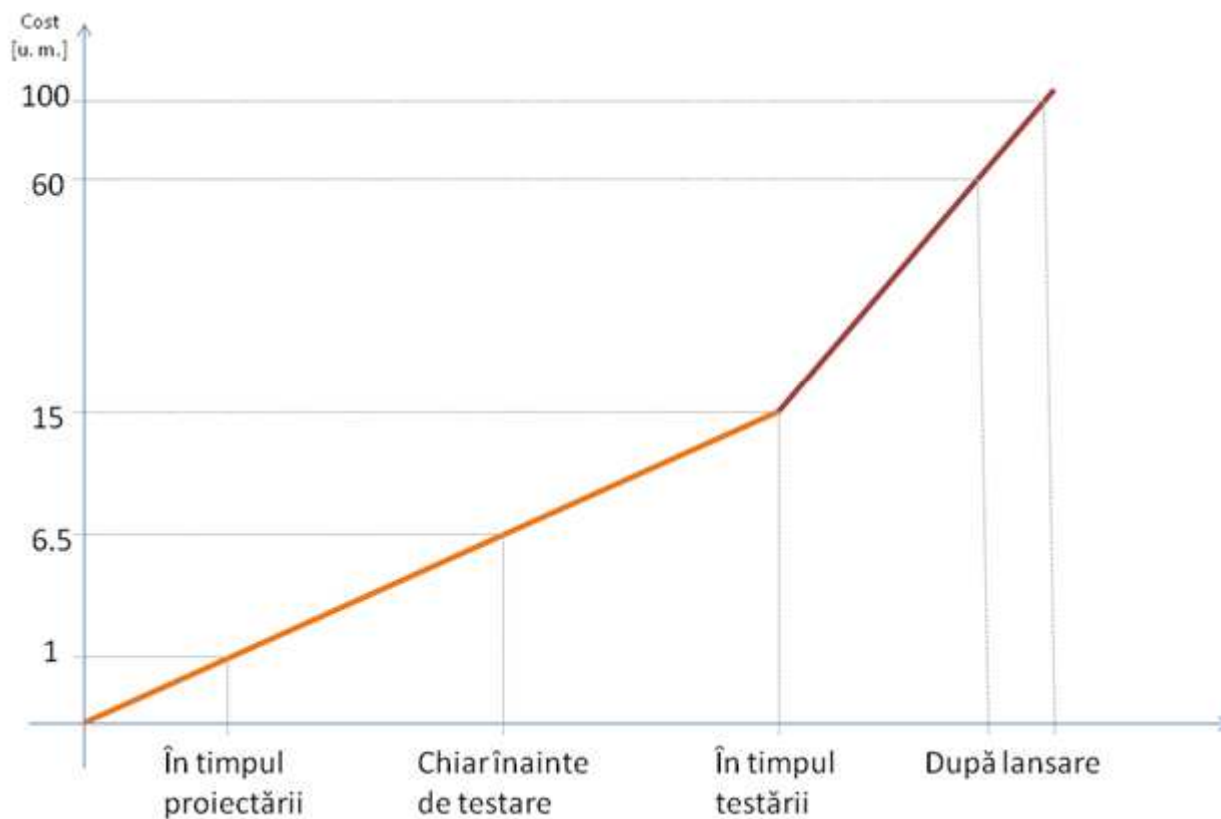


Figura 1 - Costul corectării defectului în funcție de când este descoperit

7.1. Modele de dezvoltare software – V-Modell

7.1.3. Etapele modelului V

Preluarea specificațiilor

În faza de preluare a specificațiilor (Requirements Analysis), sunt colectate cerințele sistemului cerut prin analiza nevoilor utilizatorului. Acest pas nu determină însă cum o să fie proiectat sau construit software-ul. În acest pas, utilizatorul este intervievat, și este generat un document numit "Cerințele utilizatorului". Dacă este vorba de un proiect într-un domeniu în care inginerii ce vor construi sistemul nu au cunoștințe profunde, de regulă se va utiliza o echipă mixtă alcătuită din ingineri software cât și din specialiști din domeniul respectiv. Documentul creat în urma acestui proces va descrie cerințele utilizatorului în ceea ce privește funcționarea sistemului, interfețele cu utilizatorii, performanța acestuia, datele de intrare și ieșire, măsuri de siguranță necesare, costurile în care trebuie să se încadreze sistemul precum și orice alte cerințe. Utilizatorul are obligația de a verifica acest document, el stând după aceea ca și bază în proiectarea mai departe a sistemului. Tot pe baza acestui document, în această fază, testerii pot deja proiecta metodele de testare ce se vor implementa în *testul de acceptare de către utilizator* (user acceptance test).

Cerințele sistemului

Pe baza specificațiilor primite de la utilizator se vor formula cerințele pentru sistemul dezvoltat (System Specification). Dacă în prima etapă, cerințele sistemului era formulate, astfel încât să fie clar și pentru utilizator/client și pentru proiectanți, de această dată, specificațiile vor fi formulate stric într-un limbaj tehnic, folosind termeni care să fie bine înțeleși de către inginerii proiectanți, dar care s-ar fi putut să fie ambigue pentru utilizator. Pe baza acestor specificații se formulează testele ce se vor rula în faza de *testare a sistemului* (system testing).

7.1. Modele de dezvoltare software – V-Model

7.1.3. Etapele modelului V

Proiectarea sistemului

Urmând specificațiile impuse de pasul anterior, mai departe se trece la proiectarea sistemului (System Design). În acest pas se construiește *schema bloc* a sistemului ce urmează să fie realizat, urmând ca fiecare componentă a schemei bloc să fie analizată apoi de echipe specializate. Tot în acest pas se definesc modul prin care se vor interfața componentele între ele, cum ar fi de exemplu metode sau protocoale de comunicare folosite. Din acest pas vor rezulta testele necesare pentru a *testa interfațarea*, pe partea dreaptă a modelului V.

Proiectarea componentelor

În acest pas, sistemul fiind deja descompus în elemente bloc funcționale, se vor proiecta aceste elemente (Component Design). Aici, echipe specializate pun la punct fiecare detaliu ce ține de proiectarea componentelor electronice. Se stabilesc exact funcțiile ce le va executa componenta, se construiește schema principală a acestuia, se aleg componentele ce se vor folosi la realizarea lui și se face proiectarea PCB-ului. Din metodele de proiectare a componentelor precum și din specificațiile ce definesc funcțiile acestuia se vor deduce metode pentru *testarea componentelor*.

Realizarea componentelor

Pasul de realizare a componentelor (Component Construct) este cel mai de jos pas în modelul V. Acesta este ultimul pas care se execută, înainte de a se trece la testarea sistemului. În această etapă se execută plăcutele cu cablaj imprimat și acestea se echipează conform proiectării ce s-a realizat în pasul anterior. După acest pas se începe "urcarea virtuală" pe partea dreaptă a modelului V, și începe practic procedura de testare.

7.1. Modele de dezvoltare software – V-Modell

7.1.3. Etapele modelului V

Desigur, gradul în care un sistem trebuie testat, și procedurile care se folosesc pentru testarea acestuia variază la fiecare proiect și depind de caracteristicile sistemului, precum și de modul în care sistemul a fost construit. Totuși, un principiu al testării poate fi urmat pentru toate sistemele și pașii ce trebuie urmați sunt prezentați în continuare, pe modelul V.

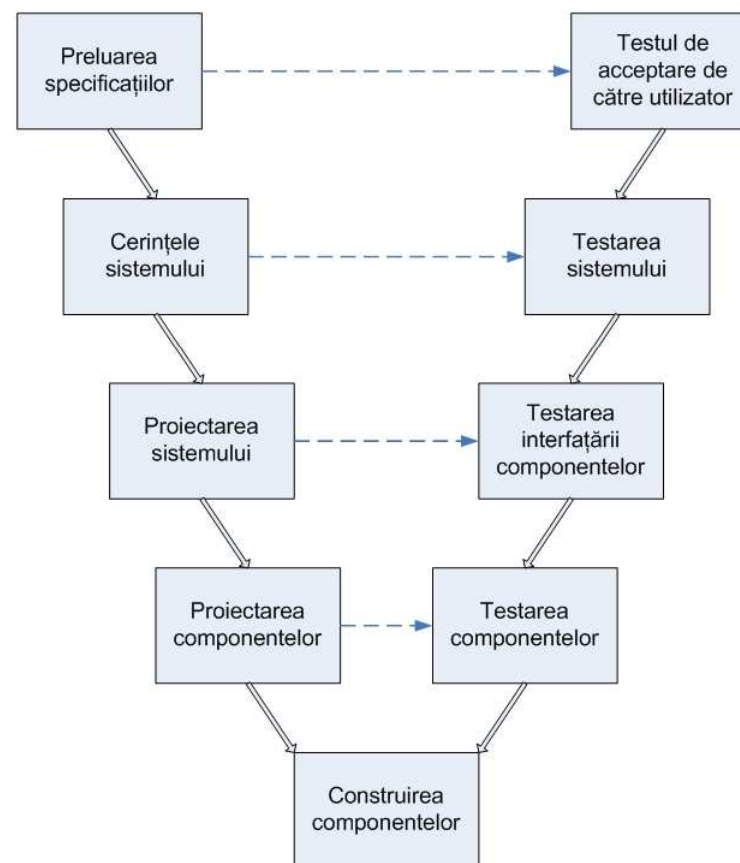


Figura 2 - Etapele modelului V

7.1. Modele de dezvoltare software – V-Model

7.1.3. Etapele modelului V

Testarea componentelor

Testarea componentelor (Component Testing) este practic primul pas de testare ce se realizează. În această fază testerii execută testare de tipul "white-box" și ei trebuie să se asigure că componenta îndeplinește toate cerințele de funcționare în condiții normale cât și în condiții de funcționare la limită. Teoretic, toate procesele de testare, deci și testarea componentelor, ar trebui să fie realizate de testeri independenți. În practică însă, de cele mai multe ori, din lipsă de personal sau de fonduri, de cele mai multe ori testarea componentelor se face de echipa care l-a dezvoltat, în timp ce se finalizează fazele de proiectare, pentru a se asigura ca componenta funcționează în mod corespunzător [4]. Acesta nu este metoda optimă de testare, însă în cele mai multe cazuri este realitatea.

Testarea interfațării componentelor

După ce componentele au fost construite, și în pasul anterior s-a asigurat că independent funcționează corect, următorul pas este de a le conecta împreună, pentru a se asigura că se interfațează conform standardelor prescrise (Interface Testing). Acest test nu este focalizat pe ce fac exact componentele ci mai degrabă cum comunică acestea împreună. Acesta verifică cum tratează componentele informațiile primite de la celelalte componente în condiții de funcționare normală, precum și cum răspund acestea la condiții de funcționare anormale, cum ar fi mesaje de eroare. Testul trebuie să fie organizat astfel încât fluxul de date să parcurgă întregul sistem, de la prima componentă până la ultima, precum trebuie testată și fiecare interfață de comunicație independent de celelalte.

7.1. Modele de dezvoltare software – V-Model

7.1.3. Etapele modelului V

Testarea sistemului

În acest pas este prima dată când întregul sistem este disponibil pentru testare ca și o entitate întreagă. Ca sistemul să fie admis, acesta trebuie să îndeplinească cerințele stabilite de specificațiile de sistem, și în această fază testarea se execută obligatoriu de o echipa de testeri independenți. Testarea trebuie să se facă atât pentru condiții de funcționare normală, cât și pentru condiții de funcționare sub stres, la limită și pentru a testa sistemele de protecție impuse, în condiții de utilizare anormală. De asemenea sistemul se va testa și peste pragul de toleranță precizat, pentru a vedea în ce condiții cedează și a stabili timpul necesar reparări sau înlocuirii acestuia, precum și eventualele posibilități de recuperare a datelor (unde este cazul). Tot în acest pas se verifică și documentația furnizată la sistem. Se verifică dacă acesta este suficient de clară și dacă tot ceea ce este descris în acesta se aplică întocmai sistemului.

7.1. Modele de dezvoltare software – V-Modell

7.1.3. Etapele modelului V

Testul de acceptare de către utilizator

Acest test ((User) Acceptance Testing – UAT), verifică sistemul în funcție de cerințele utilizatorului, dat fiind faptul ca până în acest punct toate verificările se făceau în funcție de cerințele stabilite de către echipa de ingineri la demararea proiectării. Este similar cu testarea sistemului, diferența fiind schimbarea focusului de la proiectant la utilizator. Testarea sistemului verifică dacă sistemul cerut a fost livrat. Testul de acceptare verifică dacă sistemul livrat a fost cel care a fost cerut de utilizator.

În acest punct, cei mai buni testeri sunt însăși utilizatorii care vor lucra cu noul sistem și care cunosc deja cum funcționează vechiul sistem dacă este vorba de un update, sau știu exact ce este așteptat ce la acesta dacă vorbim de un sistem complet nou. Prin urmare, pentru a face testarea, un eșantion de utilizatori este ales pentru a face testarea produsului în această fază. Această formă de testare se mai cheamă și *”testare beta”* (Beta testing) sau *”testare pe utilizatorul final”* (end user testing).

Conceptul pe care se bazează acest tip de testare este folosirea sistemului în modul în care și utilizatorul ar face-o, incluzând fiecare proces și funcționalitate a acestuia. Aici se testează folosirea sistemului în situații cu care numai utilizatorii familiarizați cu sistemul sunt capabili să le reproducă, situații pe care inginerii proiectanți nu aveau cum să le anticipeze și să le testeze. De obicei, un sistem bine definit și bine dezvoltat va trece cu brio acest gen de test.

7.1. Modele de dezvoltare software – V-Model

7.1.4. Obiectivele modelului V

Modelul V oferă ghidaj pentru planificarea și execuția proiectului. Următoarele scopuri sunt urmate să fie atinse prin utilizarea acestui model în timpul execuției:

- Minimizarea riscurilor asupra proiectului: modelul V îmbunătățește transparența proiectului și controlul acestuia. Prin urmare permite o recunoaștere timpurie a deviațiilor de la planul de proiectare, și ușurează managementul procesului
- Îmbunătățirea și garanția calității: ca și un model de proces standardizat, modelul V garantează că produsul livrat este complet și are calitatea cerută.
- Reducerea costului total pe durata întregului proiect: efortul pentru dezvoltare, producție, operare și mentenanță pentru sistem pot fi calculate și controlate într-o manieră transparentă.

7.1. Modele de dezvoltare software – V-Model

7.1.5. Neajunsurile modelului V

Desigur, ca orice model, și modelul V are anumite limitări și neajunsuri. Într-un articol publicat de firma "Claro Testing", intitulat "The Dangerous and Seductive V Model" [5], sunt prezentate majoritatea acestora.

Una dintre aceste neajunsuri este implicarea testerilor și mai ales a utilizatorilor prea târziu în proiect. Testul de acceptare de către utilizatori este executat abia pe produsul final, și prin urmare, orice neajuns este greu sau în cele mai multe cazuri chiar imposibil de corectat, fără reluarea proiectului aproape în întregime.

Tot implicarea târzie a testerilor duce și la reducerea semnificativă a timpilor acordați testării. Dacă un proiect este în întârziere, pentru a se livra produsul la termenul limită, timpii de testare vor fi scurtați față de planul original. Acesta va duce la o acoperire mai mică a testelor efectuate și prin urmare la livrarea unui sistem de calitate mai slabă.

Pentru a evita totuși aceste probleme, se propune încă de la început iterarea modelului V în mai multe etape. Astfel, se propune mai întâi construirea unui prototip al produsului care să fie testat până la testul de acceptare de utilizator, după care în cazul unor neajunsuri să fie reluat întregul model V, cu reproiectarea sistemului. Desigur, acest lucru necesită mult mai mult timp și mai mulți bani, și prin urmare, majoritatea firmelor clasează problemele raportate de utilizatorii finali ca fiind "de tip cosmetic" și nu se ocupă niciodată pe rezolvarea acestora [5].

7.1. Modele de dezvoltare software – V-Model

7.1.6. Modelul V multiplu

Modelul V multiplu vine întocmai în sprijinul ideii care susține că modelul V clasic trebuie iterat pentru fiecare stare de dezvoltarea a dispozitivului construit. Acesta se bazează pe ideea că pe durata procesului de dezvoltare diferitele stări fizice ale dispozitivului sunt produse: prima dată un model ce simulează comportamentul sistemului, apoi diferite prototipuri care evoluează apoi în produsul final. Toate aceste forme ale produsului respectă modelul V *clasic*, de unde vine denumirea de model V multiplu.

Sistemul este de obicei proiectat dintr-o secvență de prototipuri, care se apropie din ce în ce mai mult de produsul final. Mai întâi este construit un model al sistemului pe un calculator, care simulează comportamentul acestuia. După ce modelul este corect din punct de vedere funcțional, se generează cod din model, care apoi este implementat în prototip. Hardware-ul experimental al prototipului este apoi pas cu pas înlocuit cu hardware real, până când sistemul ajunge la forma sa finală, după care începe producția acestuia în masă. Motivul pentru care se folosesc acești pași intermediari în proiectare sunt faptul că a schimba prototipul este mult mai ieftin și durează mai puțin în timp decât să schimbi produsul final, și și mai ieftin e să schimbi ceva la model.

7.1. Modele de dezvoltare software – V-Model

7.1.6. Modelul V multiplu

Modelul V multiplu (Figura 3), bazat pe bine cunoscutul model V, este un model de dezvoltare care ia în considerare acest fapt. În principiu fiecare instanță a produsului (modelul, prototipurile și varianta finală) urmăresc un proces complet de dezvoltare a modelului V clasic, incluzând activitățile de dezvoltare, construire și testare. Acesta înseamnă de exemplu că funcționalitatea completă poate să fie testată atât pe modelul produsului cât și pe prototipurile sau varianta finală a acestuia. Pe de altă parte, unele proprietăți tehnice detaliate, cum ar fi impactul asupra mediului înconjurător, nu pot fi testate pe model foarte bine, și acestea trebuie să fie testate pe prototip.

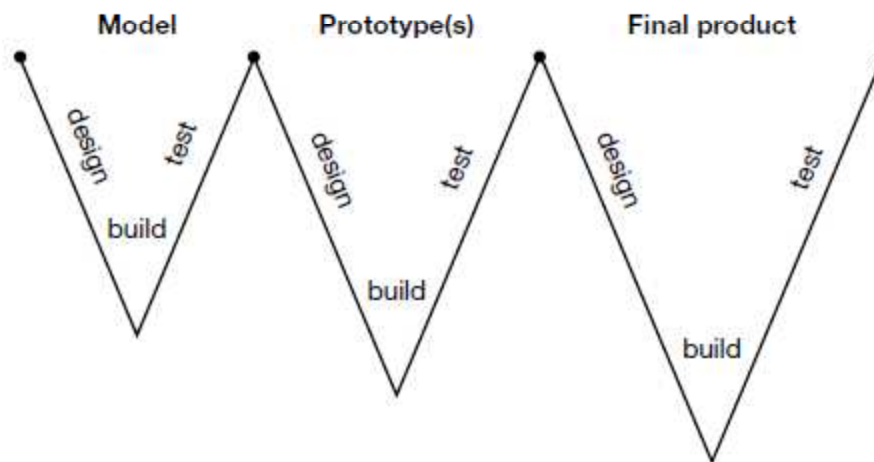


Figura 3 - Modelul V multiplu [6]

7.1. Modele de dezvoltare software – V-Modell

7.1.6. Modelul V multiplu

Procesul de testare implică o cantitate foarte mare de activități. Sunt multe tehnici de proiectare care vor fi aplicate, nivele și tipuri de test care trebuie executate, și în general, considerente legate de testare, care trebuie luate în considerare. Modelul V multiplu asistă în structurarea acestor activități. Prin alocarea lor pe ciclul V, oferă răspuns la întrebările: "Când pot activitățile să fie executate?", "Care parte a testării este cea mai relevantă în care stadiu al dezvoltării produsului?".

Organizarea testării în situația complexă a modelului V este însuși o sarcină dificilă. Deși într-adevăr detaliile sunt diferite la fiecare proiect în parte, se pot aplica însă aceleași principii pentru a structura activitățile de testare.

Mai departe sunt prezentate aranjarea diferitelor activități de testare în funcție de stadiul de dezvoltare în care se afla produsul.

7.1. Modele de dezvoltare software – V-Modell

7.1.6. Modelul V multiplu

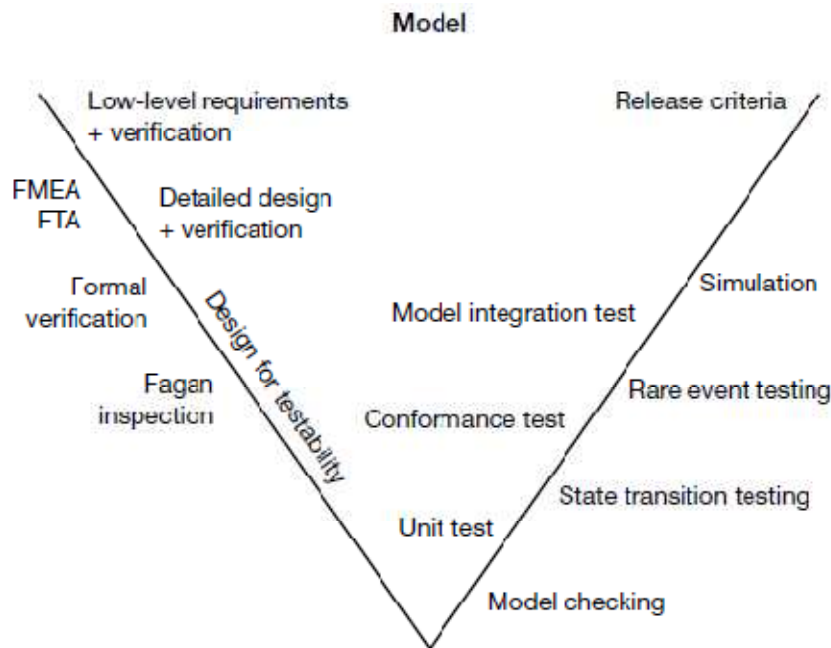


Figura 4 - Modelul V pentru stadiul de model al produsului [6]

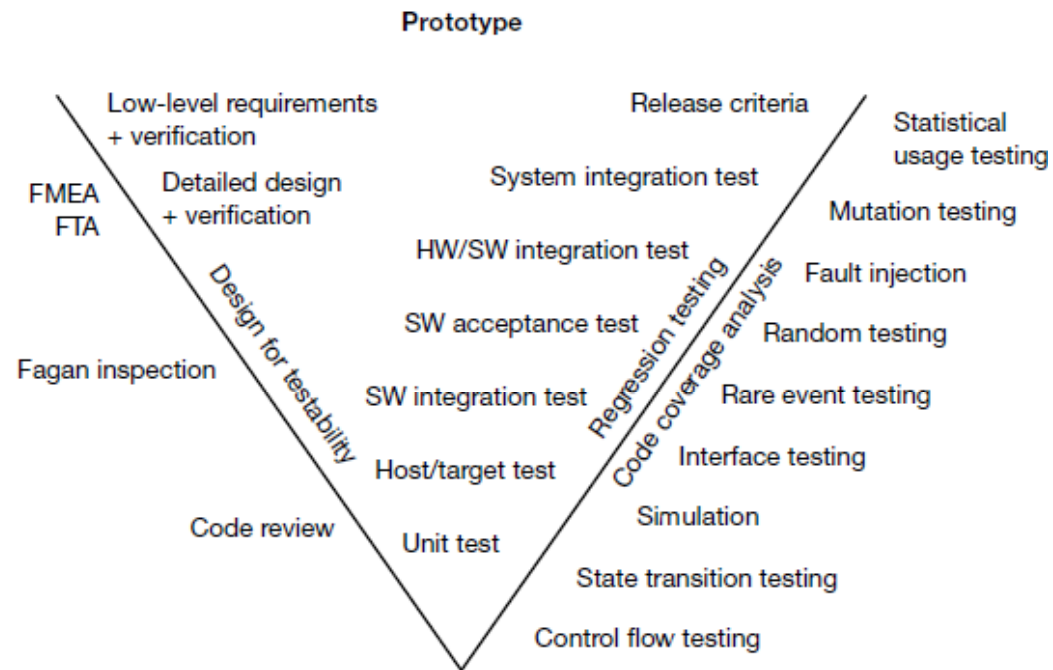


Figura 5 - Modelul V pentru stadiul de prototip al produsului [6]

7.1. Modele de dezvoltare software – V-Modell

7.1.6. Modelul V multiplu

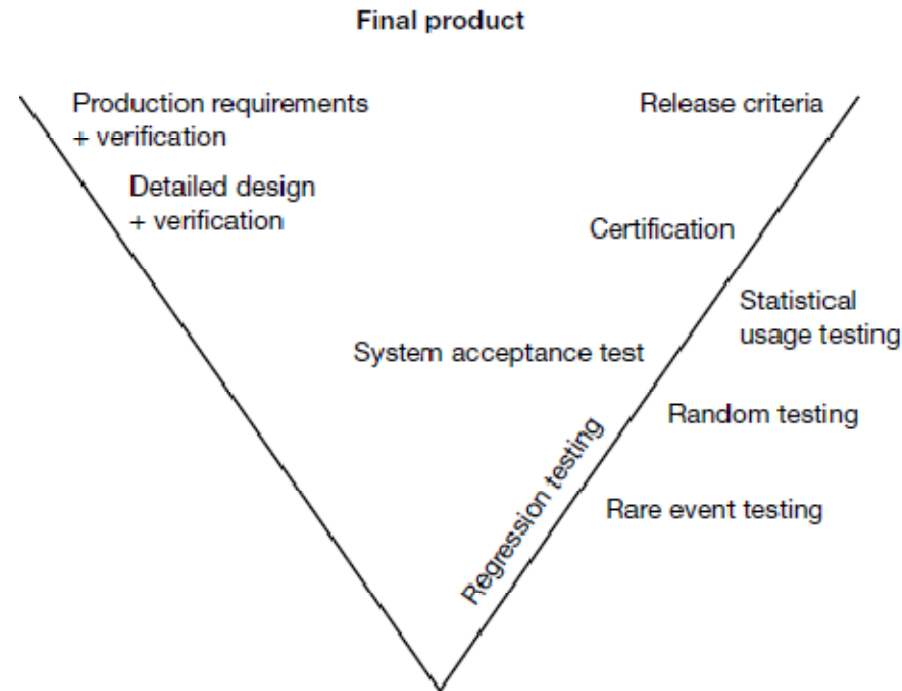


Figura 6 - Modelul V pentru stadiul final al produsului [6]

7.1. Modele de dezvoltare software – V-Modell

7.1.6. Modelul V multiplu

Modelul V multiplu în cei trei ciclii secvențiali V nu ia în considerare obiceiul de a descompune un sistem complex în unități funcționale mai simple. Dezvoltarea unui asemenea sistem începe prin stabilirea specificațiilor de proiectare la nivel înalt, urmat de o fază de planificare la nivelul arhitecturii, unde se determină ce componente (hardware și software) sunt necesare pentru realizarea proiectului. Aceste componente sunt mai departe dezvoltate separat, în final fiind integrate într-un sistem complet. Modelul V simplu poate fi aplicat acestui proces de dezvoltare la nivel înalt. Partea stângă a modelului se ocupă de descompunerea sistemului în componentele sale. Partea din mijloc a modelului V constă în ciclii de dezvoltare paralele pentru toate componentele, iar partea din dreapta a modelului V se ocupă de integrarea componentelor în sistem (Figura 7).

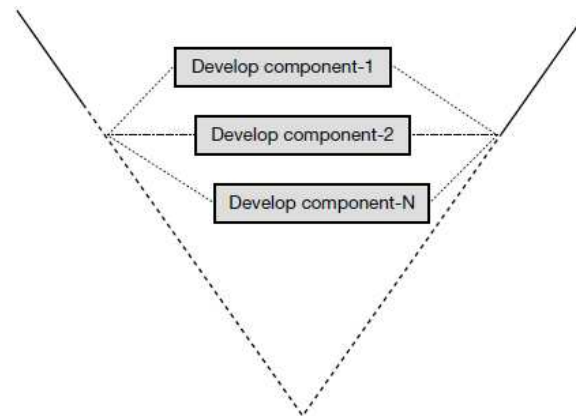


Figura 7 - Model V multiplu cu dezvoltarea paralelă a componentelor [6]

7.1. Modele de dezvoltare software – V-Modell

7.1.6. Modelul V multiplu

De fapt, modelul V pur secvențial este aplicabil numai la nivelul componentelor. De obicei nu sistemul complet se modelează întâi, apoi construit prototipul lui ș.a.m.d. Componentele sunt cele care sunt construite în pași așa. Acest fapt explică de ce unele activități de dezvoltare și unele probleme ce ar putea apărea nu pot fi bine plasate pe cele 3 V-uri ale modelului V, de exemplu cerințele la nivelul înalt și la nivelul inferior, planul de siguranță, și instrumentele specifice construirii. Acest lucru se explică prin faptul că acestea aparțin procesului global de dezvoltare.

Când modelul V la nivelul sistemului se combină cu modelul multiplu V la nivelul componentelor, rezulta așa numitul "model V încorporat" (Figura 8).

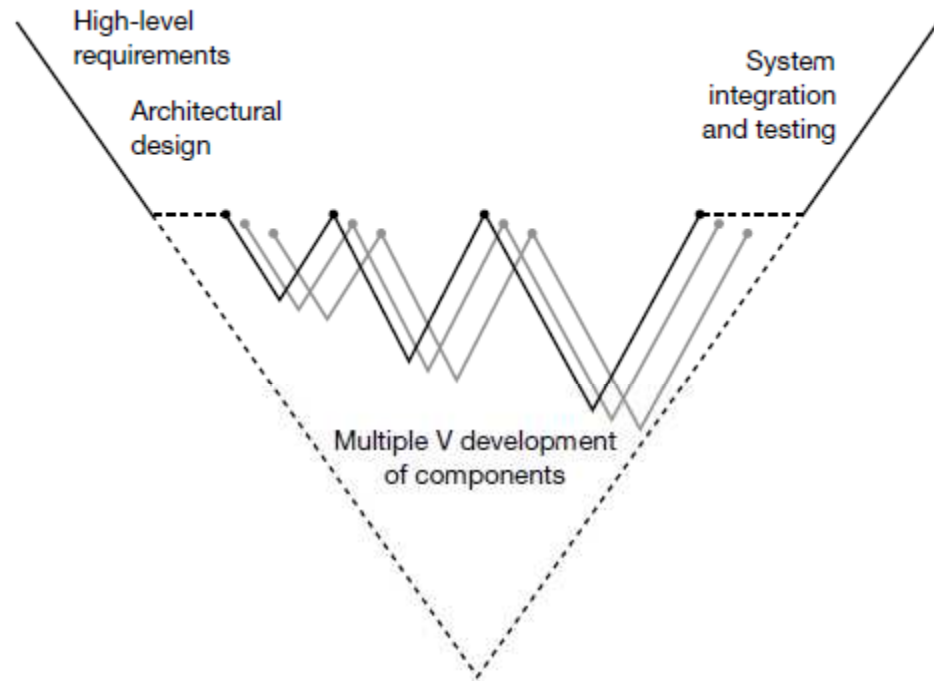


Figura 8 - Model V încorporat [6]

7.1. Modele de dezvoltare software – V-Modell

7.1.6. Modelul V multiplu

Cu acest model, toate activitățile legate de testare pot fi alocate nivelului corespunzător, la locul corect. Modelul V multiplu nu este util numai când trebuie proiectate și executate procedeele de testare pentru un produs complet nou, dar și în cazul lansării unei versiuni noi ai aceluiași produs.

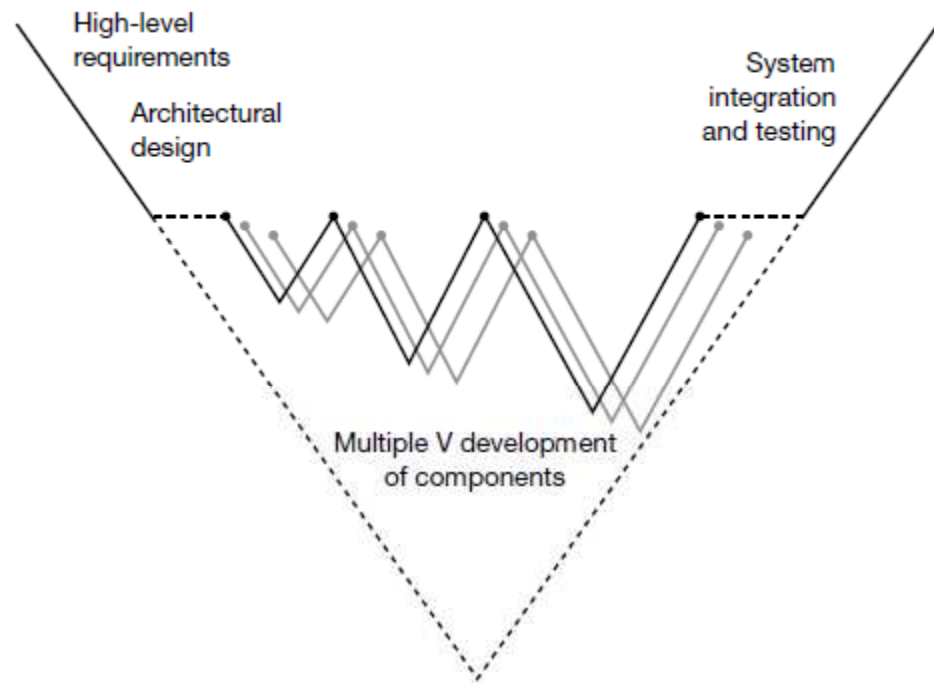


Figura 8 - Model V încorporat [6]

7.2. Reguli de dezvoltare software

7.2.1. Dezvoltarea aplicatiilor software – vedere generala

- **Buclo de control simpla** – software-ul are o singura bucla care cheama subrutine, fiecare subrutina manageriaza o parte din hard sau soft
- **Sistem controlat de intreruperi** – task-urile efectuate de catre sistem sunt chemate de diferite tipuri de evenimente (ex. Extern, timer, UART, etc.)
- **Multitasking**
 - Nonpreemptive - similar cu bucla de control simpla
 - Preemptive – are nevoie de un “programator” (eng. scheduler) pentru task-uri.
- **Microkernel**
 - Contine o simpla abstractizare a hardware-ului, avand un set primitiv the rutine ce implementeaza o parte minimala a uni OS (managementul memoriei, multitasking si comunicari interproces).
- **Kernele monolitice**
 - Toate serviciile OS ruleaza in kernel

7.2. Reguli de dezvoltare software

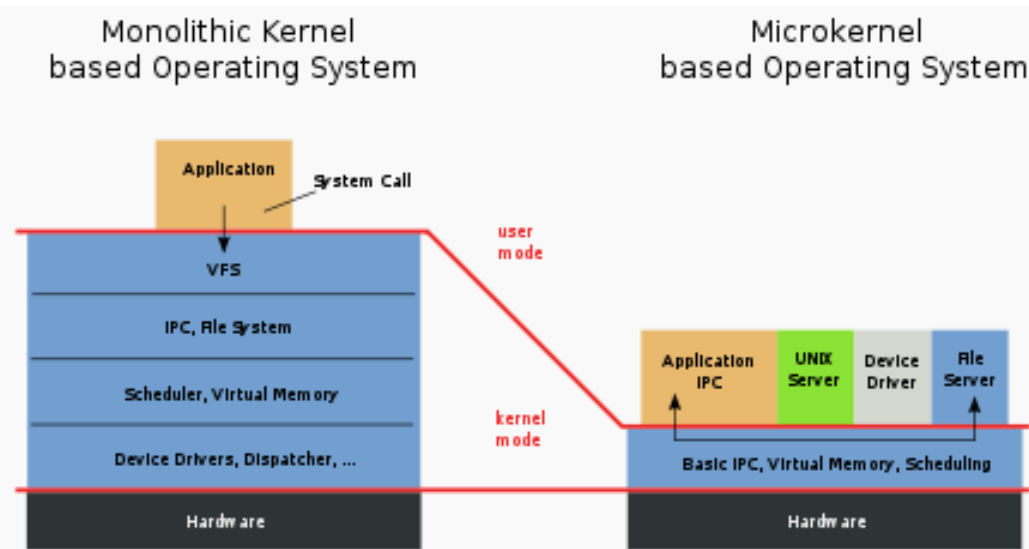
7.2.1. Dezvoltarea aplicatiilor software – vedere generala

- **Microkernel**

- Contine o simpla abstractizare a hardware-ului, avand un set primitiv the rutine ce implementeaza o parte minimala a uni OS (managementul memoriei, multitasking si comunicari interprocese).

- **Kernele monolitice**

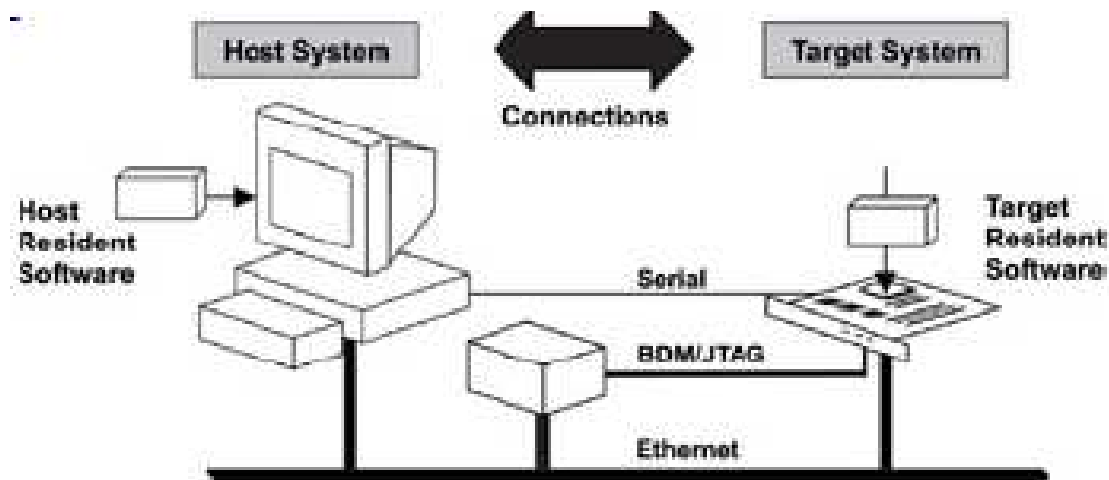
- Toate serviciile OS ruleaza in kernel



7.2. Reguli de dezvoltare software

7.2.1. Dezvoltarea aplicatiilor software – vedere generala

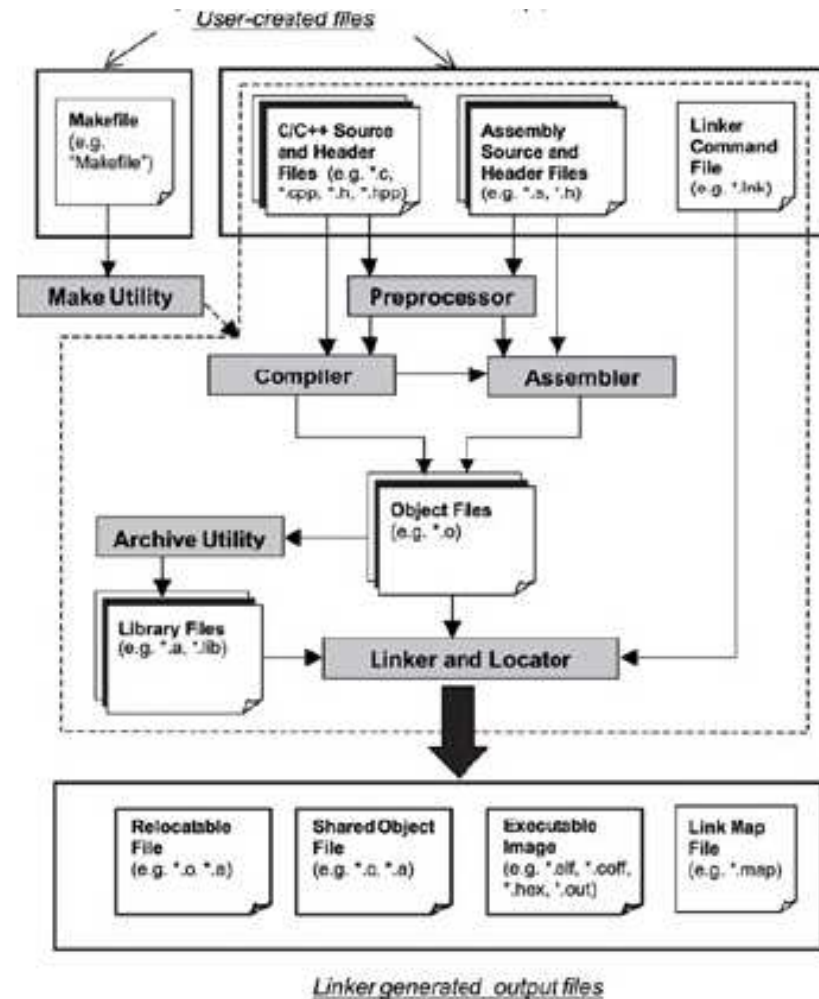
Dezvoltarea software se face folosind metodologia **cross-platform** :



7.2. Reguli de dezvoltare software

7.2.1. Dezvoltarea aplicatiilor software – vedere generala

Tool-urile oferite de sistemul “gazda” sunt: cross compiler, linker, source-level debugger
Sistemul tinta (sistmul embedded) contine: loader dinamic, monitor si un agent pentru debug.
Aplicatiile trebuie mai intai dezvoltate, apoi compilate in cod obiect si apoi linkedate in programe executabile pentru sistemul embedded.

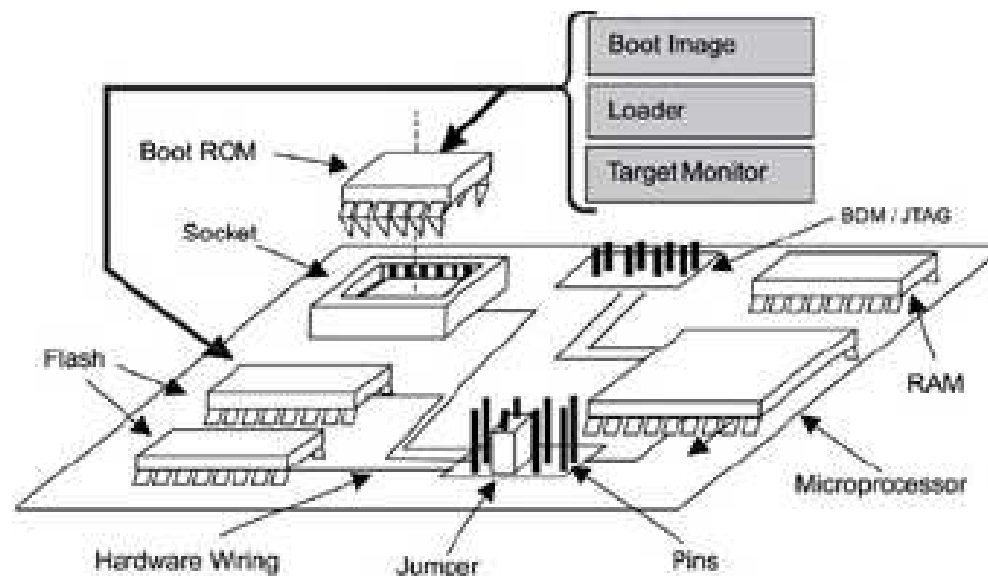


7.2. Reguli de dezvoltare software

7.2.2. Initializarea aplicatiilor software – vedere generala

O imagine executabila construita pentru sistemul emebedded poate fi transferata prin:

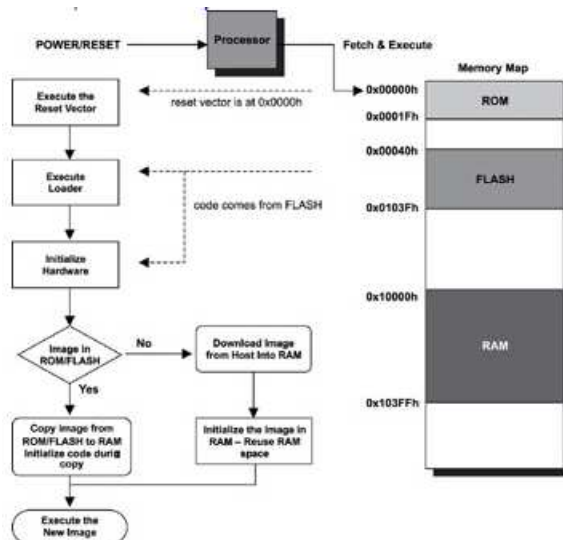
1. Programarea intregii imagini in EEPROM sau fmemoria flash
2. Descarcarea imaginii prin conexiune seriala (RS232), CAN etc.
3. Descarcarea imaginii prin interfete JTAG sau BDM



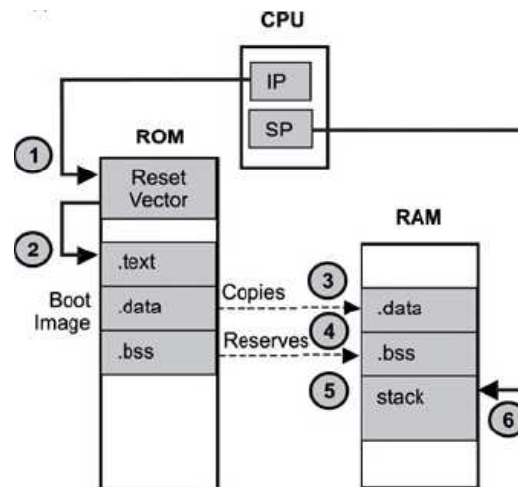
7.2. Reguli de dezvoltare software

7.2.3. Metode de bootare pentru aplicatii software – vedere generala

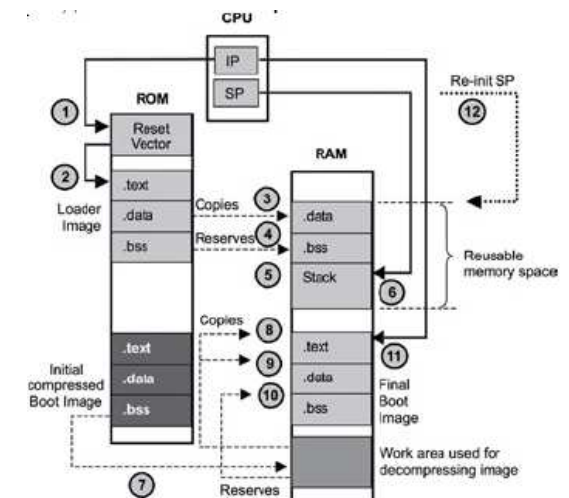
Metoda bootstrap



Rularea codului din ROM
folosind date din RAM



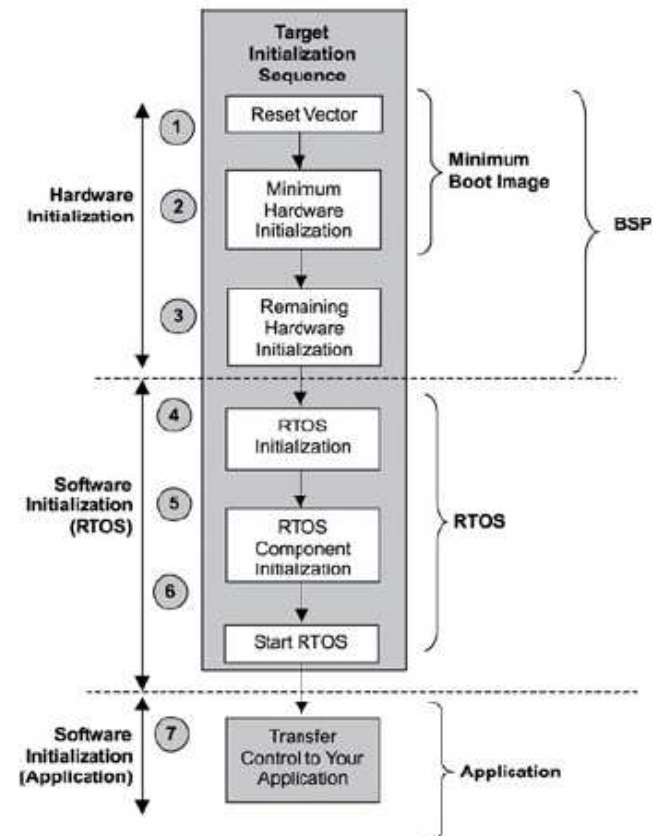
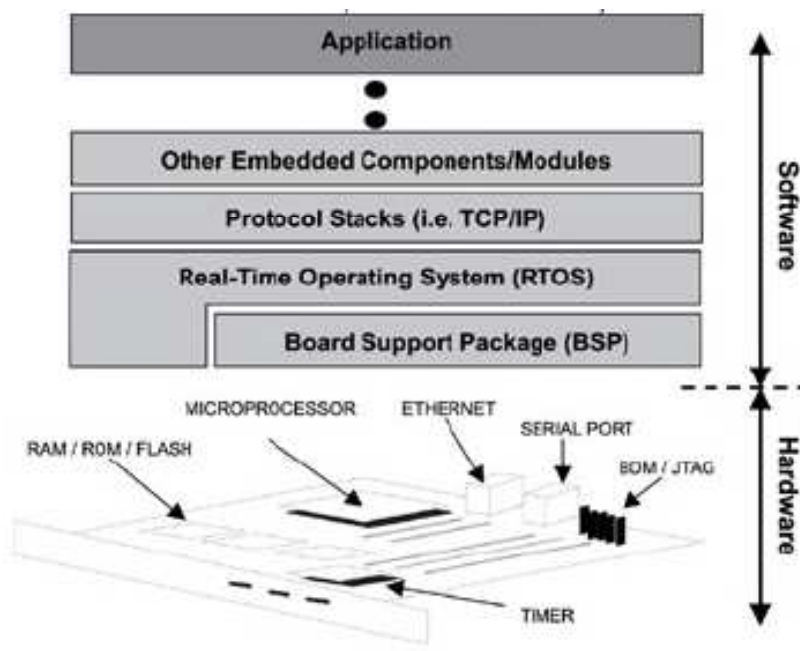
Rularea din RAM dupa
trasferul imaginii din
ROM



7.2. Reguli de dezvoltare software

7.2.4. Initializarea aplicatiilor software – detalii

Principalii pasi in initializarea software-ului sistemelor embedded sunt: initializarea hard, initializarea OS sau RTOS – daca exista si initializarea aplicatiei.



7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Mesajele pot fi transmise/receptionate de catre taskuri sau ISR. Mesajele sunt trimise catre *sending message objects* si receptionate de la *receiving message objects*. Identificarea mesajelor obiect se face folosind identificatori de mesaj, identificatori asignati la generarea sistemului. Catre acelasi *sending message objects* pot fi trasmise mai multe mesaje. Un *receiving message objects* primeste mesaje de la un singure mesaj obiect (cazul comunicatiei interne) sau un singur I-PDU. Un *receiving message objects* pote fi definit ca si *queued*(poate fi citit o singura data dupa care este sters) sau *unqueued*(poate fi citit de mai multe ori).

Un mesaj extern poate avea una din urmatoarele doua proprietati:

- *Triggered Trasfer Property* – mesajul in IPDU-ul corespunzator asignat este updatat si o cerere de trasmisie catre I-PDU este fauta.
- *Pending Trasfer Property* - mesajul in IPDU-il corespunzator este updatat fara o cerere de trasmisie.

Mesajele interne nu au proprietati de transfer. Ele sunt rutate instantaneu catre partea de iesire.

Exista trei moduri de trasmisie petru I-PDU:

- modul de trasmisie direct – trasmisia initializata explicit prin trasmiterea unui mesaj cu proprietatea *Triggered Trasfer Property*
- modul de trasmisie periodic – trasmisie cu o perioada presetata
- modul de trasmisie mixt – combinatie intre primele doua moduri

7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Numai pentru comunicatia externa:

- Receptia incepe cu un indicator de receptionare. Daca acest indicator nu contine o eroare mesajul este copiat in I-PDU.
- Deadline Monitoring poate invoca o eroare de receptionare a mesajului.
- Mesajul este desfacut – conversie din reprezentarea la nivelul retelei la reprezentarea la nivelul CPU

Atat pentru comunicatia interna cat si externa

- Filtrare aplicata continutului mesajelor (in unele cazuri).
- Daca nu se foloseste filtrarea mesajelor, atunci continutul este copiat in mesajul obiect corespunzator
- Daca se foloseste filtrarea, o notificare de receptionare este invocata. Notificarea este per obiect mesaj.
- Datele din mesajul obiect sunt copiate in mesajele aplicatiei.

Filtrarea – inlaturarea anumitor mesaje cand anumite conditii, setate in filtrul de mesaje, nu sunt indeplinite. Filtrarea se face per mesaj obiect. Atributele ce pot fi folosite in implementarea algoritmilor de filtrare sunt:

- Valoarea noua
- Valoarea veche
- Masca,min, max, perioada, offset, etc.
- Numar de receptionari

7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

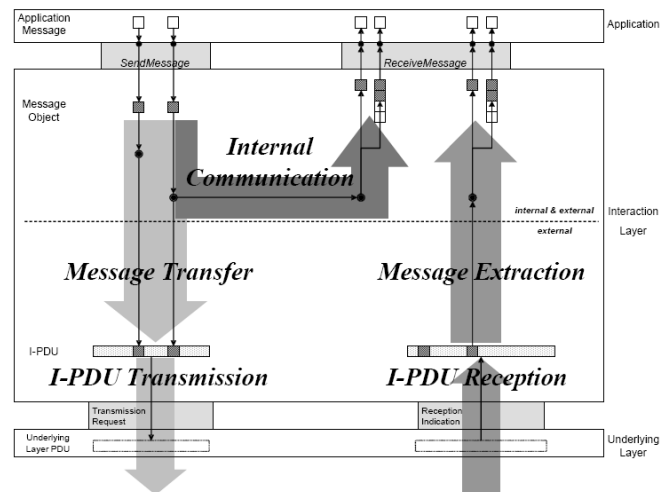
Trasmiterea unui mesaj implica transferul datelor de la nivelul aplicatiei catre I-PDU (pentru comunicatia externa) si/sau *receiving message object(s)* (comunicatie interna).

Numai pentru comunicatia externa:

- Filtare
- CPU order Message Callout
- Conversie intre reprezentari
- Network-order Message Callout – mesajul este memorat in I-PDU

Moduri de trasmsie:

1. Modul direct
2. Modul periodic
3. Modul mixt

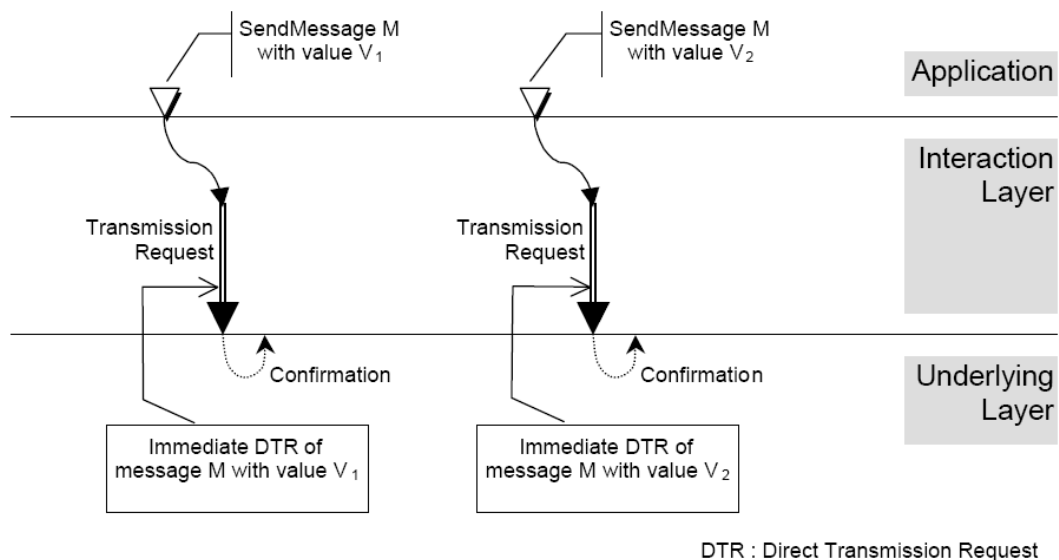
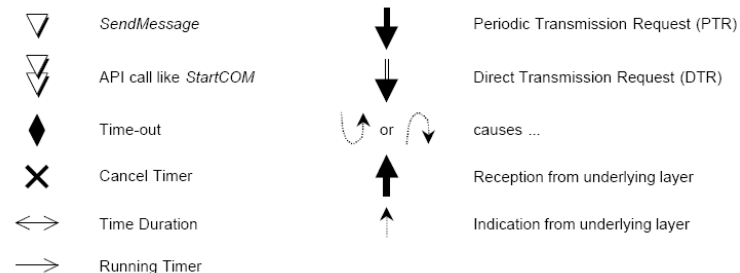


7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Moduri de trasmsie:

1. Modul direct

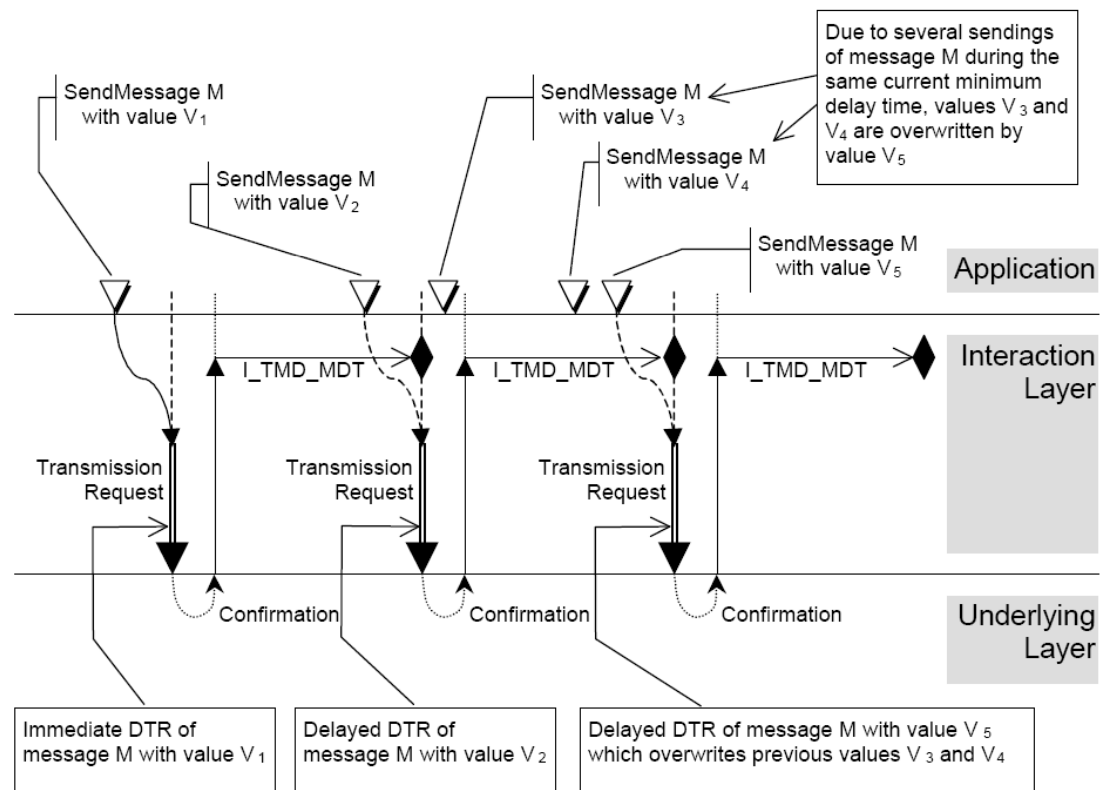
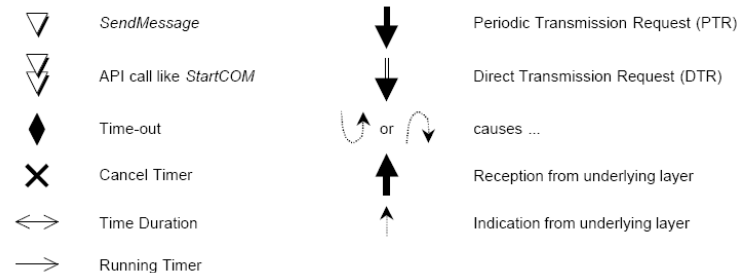


7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Moduri de trasmsie:

1. Modul direct

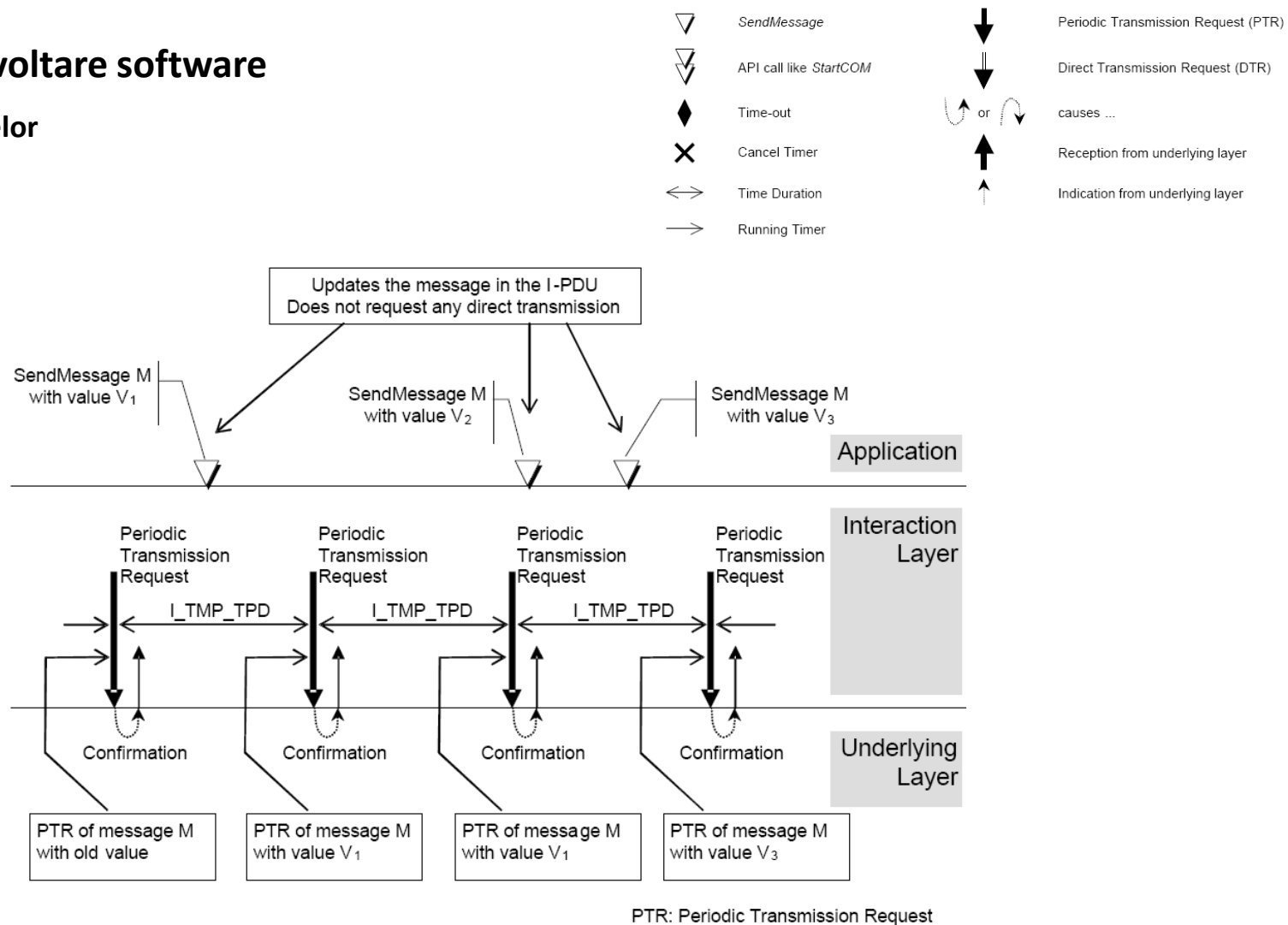


7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Moduri de trasmsie:

2. Modul periodic

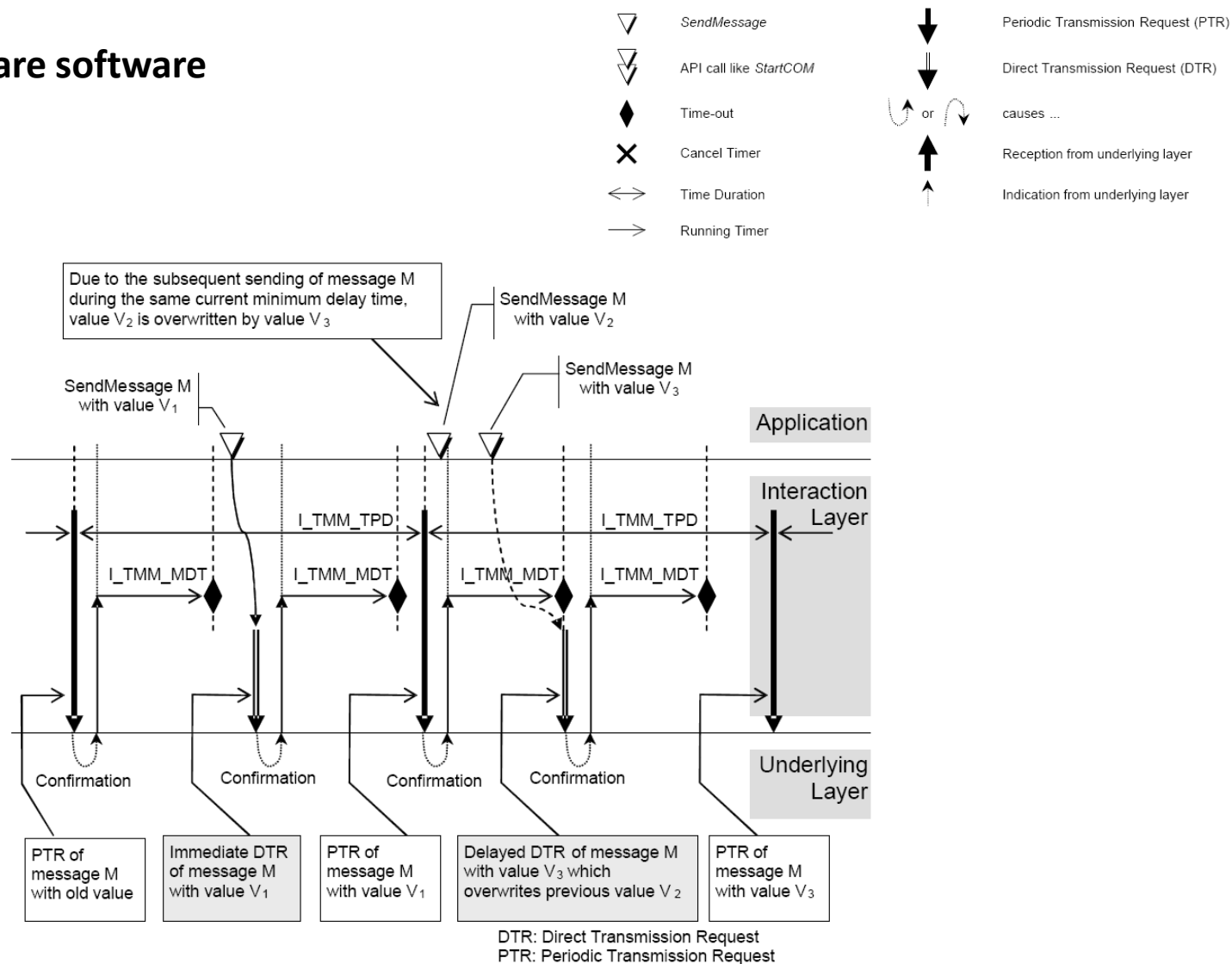


7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Moduri de trasmsie:

3. Modul mixt

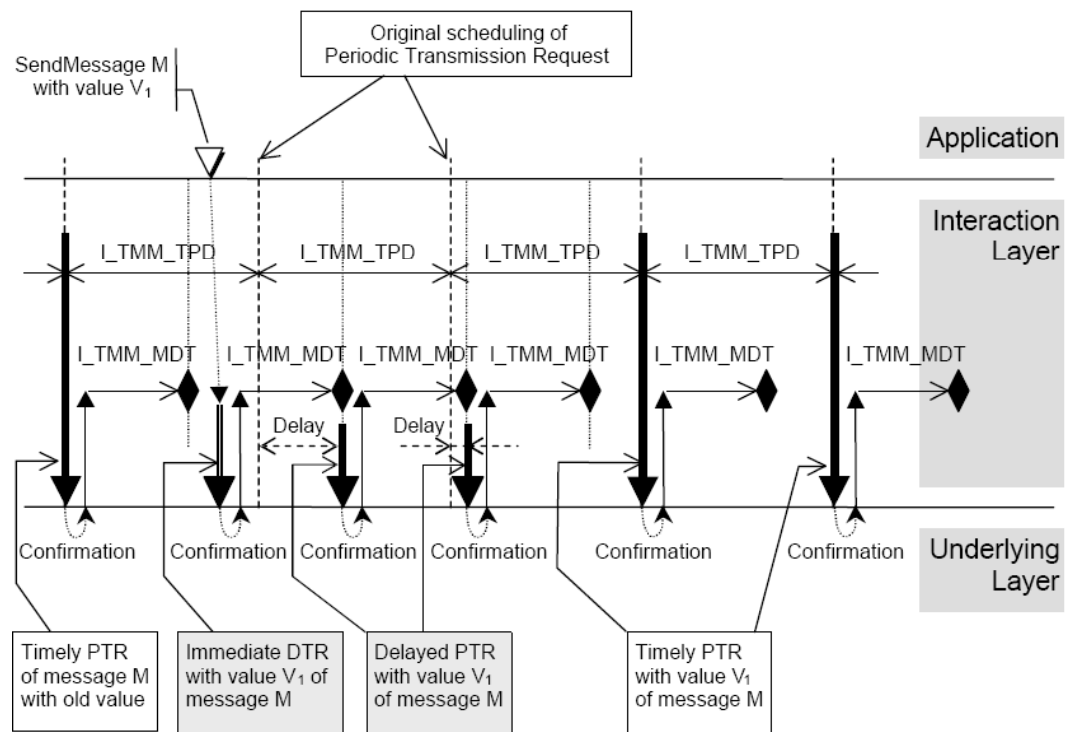
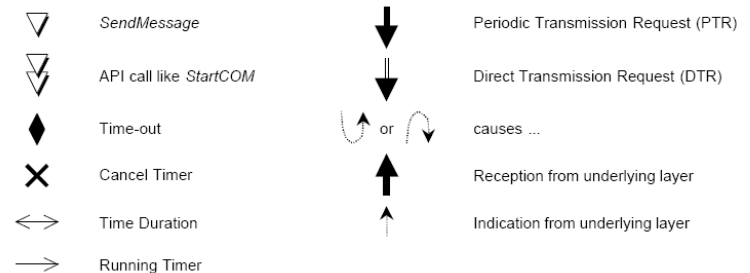


7.2. Reguli de dezvoltare software

7.2.5. Utilizarea mesajelor

Moduri de trasmsie:

3. Modul mixt



DTR: Direct Transmission Request
PTR: Periodic Transmission Request

7.2. Reguli de dezvoltare software

7.2.6. Conversii

Little-endian byte order

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	7	6	5	4	3	2	1	0
Byte 1	15 ← 2	14 ← 1	13 ← 0 LSB	12	11	10	9	8
Byte 2	23 ← 10	22 ← 9	21 ← 8	20 ← 7	19 ← 6	18 ← 5	17 ← 4	16 ← 3
Byte 3	31	30	29	28	27	26	25	24 ← 11 MSB
Byte 4	39	38	37	36	35	34	33	32

Big-endian byte order

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	7	6	5	4	3	2	1	0
Byte 1	15	14	13 ← 11 MSB	12 ← 10	11 ← 9	10 ← 8	9 ← 7	8 ← 6
Byte 2	23 ← 5	22 ← 4	21 ← 3	20 ← 2	19 ← 1	18 ← 0 LSB	17	16
Byte 3	31	30	29	28	27	26	25	24
Byte 4	39	38	37	36	35	34	33	32

7.3. Standarde de dezvoltare software

7.3.1. Standardul OSEK – vedere generala

Standardul OSEK/VDX – este o combinatie de standarde pentru sistemele embedded din industria de automobile dezvoltate de doua consortii separate care in final au fuzionat.

OSEK – germana: *Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen*

engleza: *Open systems and the Corresponding interfaces for Automotive electronics*

VDX - Vehicle Distributed eXecutive

Standardul cuprinde trei parti:

1. Operating System
2. Communication stack
3. Network Management

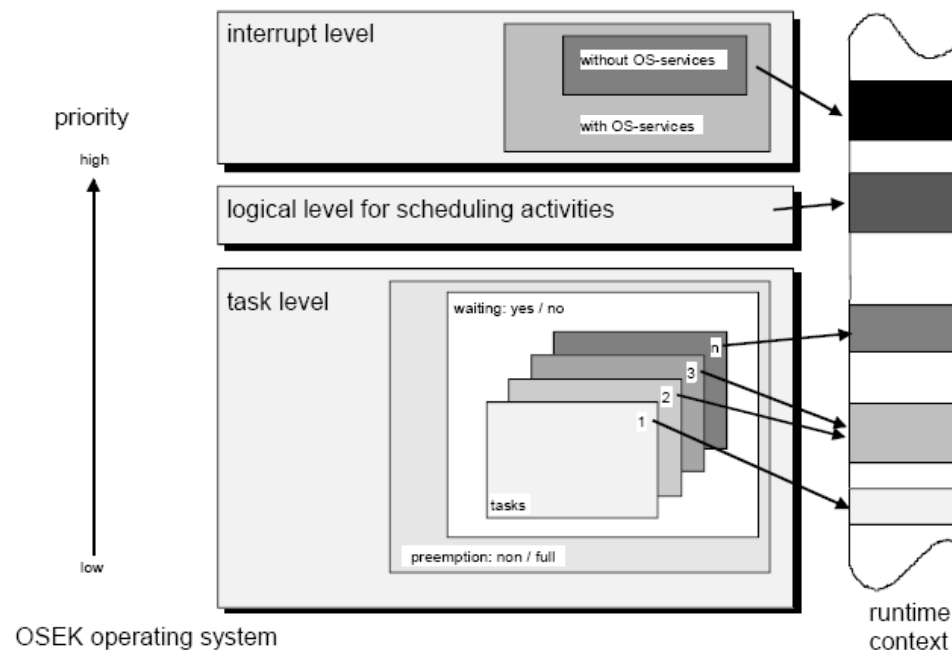
Principalul scop al OSEK este imbunatatirea portabilitatii si refolosirii softwarelului la nivelul aplicatiilor. In acest scop OSEK defineste un limbaj pentru o standardizare a informatiilor de configurare (OIL – OSEK Implementation Language)

7.3. Standarde de dezvoltare software

7.3.2. Standardul OSEK – nivele de procesare

Sistemul de operare OSEK servește ca și baza pentru aplicații independente între ele, creând un mediu ce rulează pe un procesor. OSEK OS permite controlul unui mediu de execuție real-time unde diferite procese rulează în paralel. OSEK OS furnizează diferite interfețe pentru user, aceste interfețe fiind folosite de entități ce “concurează” pentru CPU. Există două tipuri de entități: ISR (Interrupt Service Routines) și Taskuri.

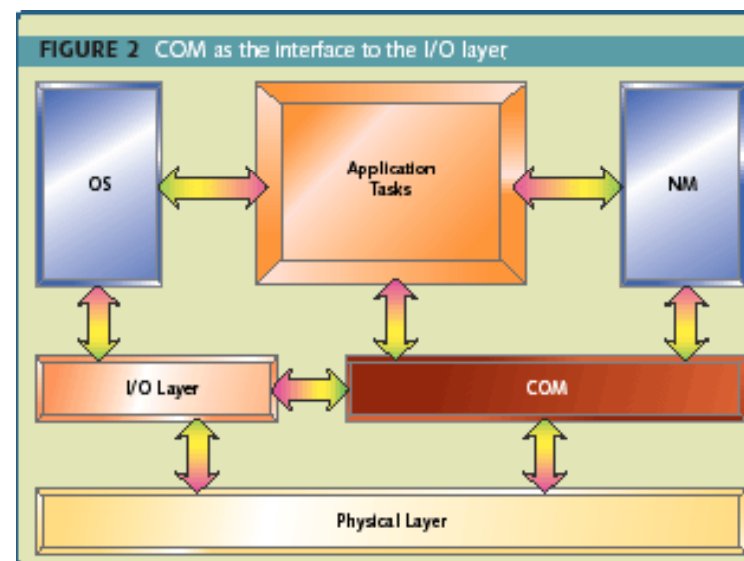
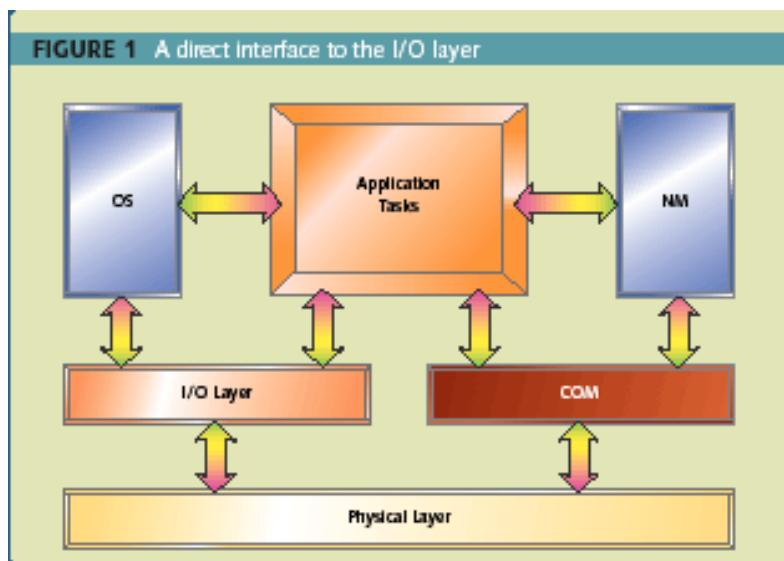
OSEK definește trei nivele de procesare: nivelul întreruperi, nivelul logic pentru scheduler și nivelul task.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

Arhitectura pe care un sistem de operare OSEK este bazata poate avea diferite forme. Doua forme frecvent utilizate sunt cele din figura 1 si 2. Diferenta dintre cele doua forme este modul in care aplicatia acceseaza interfata hardware.

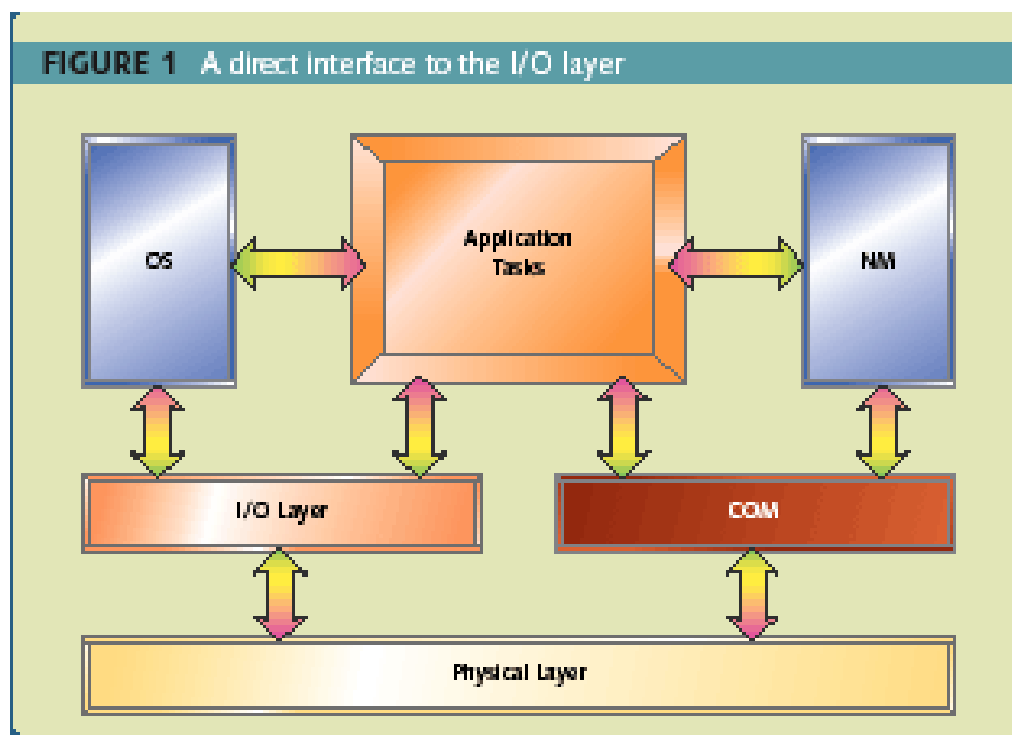


7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

În prima formă, aplicația adresează nivelul I/O direct. Aceasta parte nu este definită în standardul OSEK datorită varietății cerințelor diferitelor aplicații.

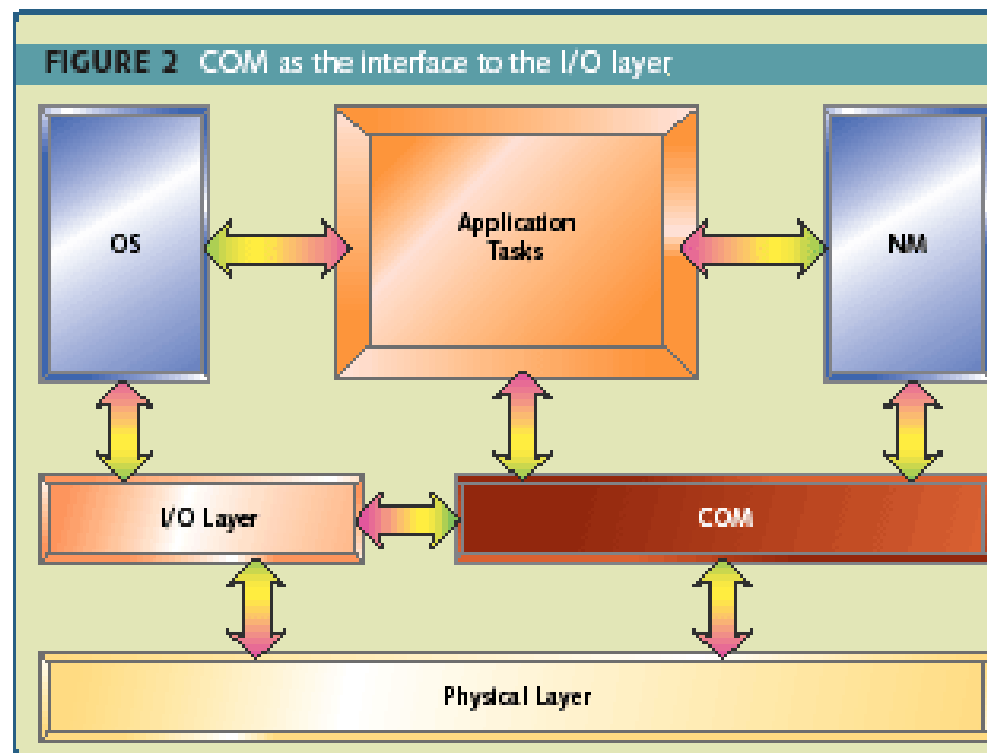
Avantajul acestei forme de implementare este răspunsul rapid la o cerere de accesare a I/O din partea task-urilor. Dezavantajul major este portabilitatea task-urilor.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

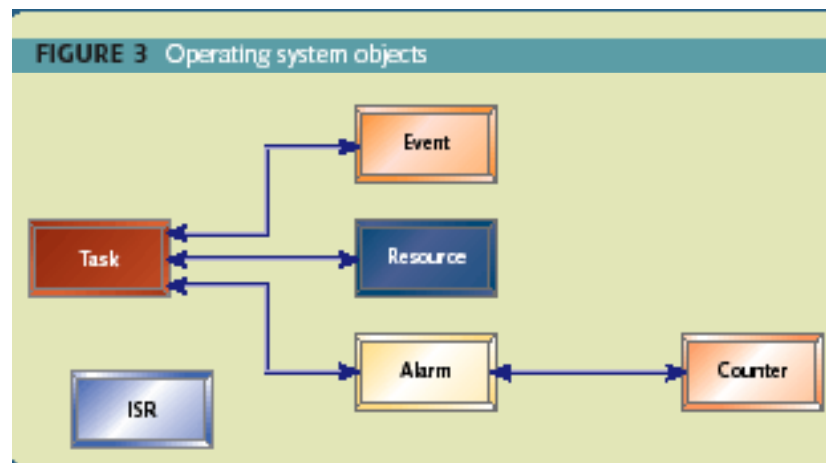
A doua forma trateaza nivelul I/O ca si un task. Fiecare aplicatie cere sau trimite informatii catre modulul COM.
Avantajul major este portabilitatea.
Dezavantajul major consta in cresterea timpului de acces la I/O.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

Sistemul de operare este prima componenta din standardul OSEK. El este compus dintr-o serie de obiecte cum sunt: task-uri, evenimente, resurse management, alarme, counters, ISR (Interrupt Service Routine). Deasemenea OS beneficiaza de implementari pentru tratarea erorilor si “hooks” pentru functiile definite de user.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

Task-urile pot fi de mai multe tipuri: basic (BT) sau extended (ET), preemptive sau non-preemptive. Taskurile externe sunt taskuri basic ce pot reactiona la evenimente externe asincrone. (pot intra in starea de wait asteptand un eveniment). BT ruleaza pana la sfarsit, in cazul in care nu sunt intrerupte. Fiecare task din sistem are asignata o prioritate fixa (asignare statica in momentul compilarii), iar scheduler-ul selecteaza intotdeauna prioritatea cea mai marea din lista de asteptarea a task-urilor pregatite pentru executie. Taskurile preemptive pot fi intrerupte de un task de prioritate mai mare atunci cand el este gata de rulare sau de o intrerupere (ISR). Taskurile non-preemptive pot fi intrerupte numai de ISR (in cazul in care ISR nu sunt disable).

Pentru a pastra conformitatea cu standardul OSEK de obicei trebuie respectate mai multe clase de conformitate. Clasele de conformitate ne ajuta la implementarea partiala a standardului.

Standardul defineste urmatoarele clase de conformitate:

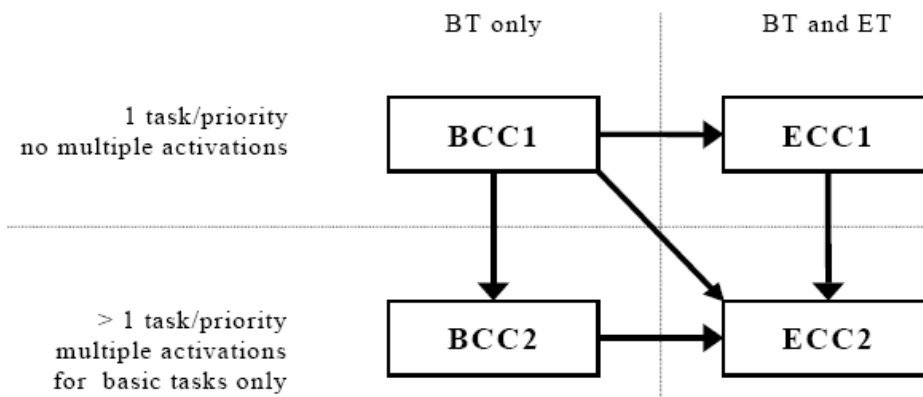
1. **BCC1** – Numai taskuri basic limitate la o singura activare/task, un task/prioritate, toate taskurile avnd prioritati diferite.
2. **BCC2** – BCC1+ mai mult de o cerere de activare/task si mai mult de un.
3. **ECC1** - BCC1, plus extended tasks.
4. **ECC2** – ECC1+mai mult de un task/prioritate si multiple cereri de activare a taskurilor pentru BT.

7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

Standardul defineste urmatoarele clase de conformitate:

1. **BCC1** – Numai taskuri basik limitate la o singura activare/task, un task/prioritate, toate taskurile avnd prioritati diferite.
2. **BCC2** – BCC1+ mai mult de o cerere de activare/task si mai mult de un.
3. **ECC1** - BCC1, plus extended tasks.
4. **ECC2** – ECC1+mai mult de un task/prioritate si multiple cereri de activare a taskurilor pentru BT.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

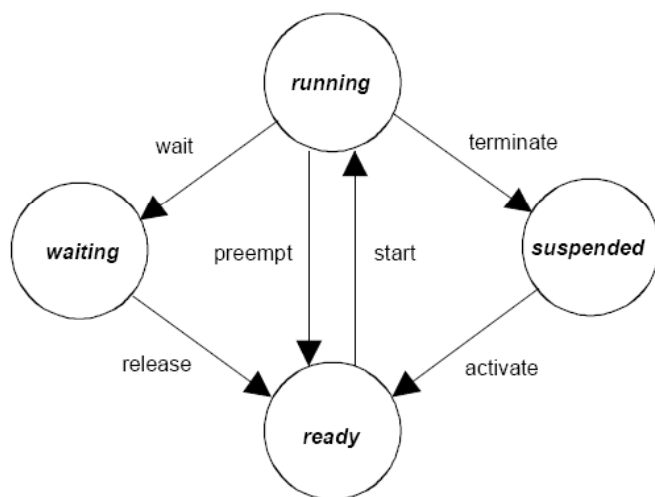
Portabilitatea aplicatiilor pote fi asumata daca minimul de cerinte sunt satisfacute. Cerintele minime pentru clasele de conformitate sunt:

	BCC1	BCC2	ECC1	ECC2
Multiple requesting of task activation	no	yes	BT ³ : no ET: no	BT: yes ET: no
Number of tasks which are not in the <i>suspended</i> state	8		16 (any combination of BT/ET)	
More than one task per priority	no	yes	no (both BT/ET)	yes (both BT/ET)
Number of events per task	—		8	
Number of task priorities	8		16	
Resources	RES_SCHEDULER	8 (including RES_SCHEDULER)		
Internal resources	2			
Alarm	1			
Application Mode	1			

7.3. Standarde de dezvoltare software

7.3.4. Standardul OSEK – task-uri

Taskurile extinse au patru stari: running, ready, waiting, suspended.

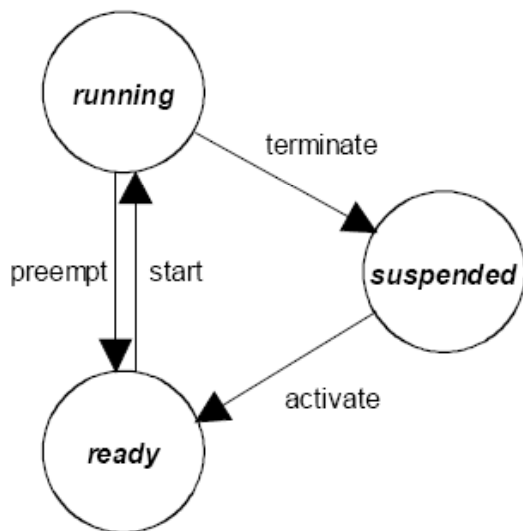


Transition	Former state	New state	Description
activate	suspended	ready	A new task is set into the <i>ready</i> state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	ready	running	A <i>ready</i> task selected by the scheduler is executed.
wait	running	waiting	The transition into the waiting state is caused by a system service. To be able to continue operation, the <i>waiting</i> task requires an event.
release	waiting	ready	At least one event has occurred which a task has <i>waited</i> for.
preempt	running	ready	The scheduler decides to start another task. The <i>running</i> task is put into the <i>ready</i> state.
terminate	running	suspended	The <i>running</i> task causes its transition into the <i>suspended</i> state by a system service.

7.3. Standarde de dezvoltare software

7.3.4. Standardul OSEK – task-uri

Taskurile basic au un model asemenator cu cel al taskurilor extinse, exceptia constand in faptul ca ele nu au starea wait. Starile tasurilor basic sunt: running, ready, suspended.



Transition	Former state	New state	Description
activate	<i>suspended</i>	<i>ready</i> ⁴	A new task is set into the <i>ready</i> state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	<i>ready</i>	<i>running</i>	A <i>ready</i> task selected by the scheduler is executed.
preempt	<i>running</i>	<i>ready</i>	The scheduler decides to start another task. The <i>running</i> task is put into the <i>ready</i> state.
terminate	<i>running</i>	<i>suspended</i>	The <i>running</i> task causes its transition into the <i>suspended</i> state by a system service.

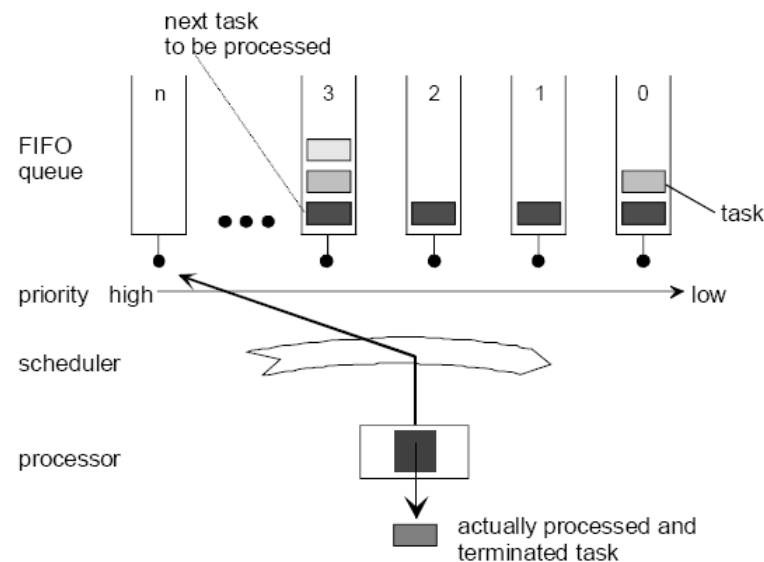
7.3. Standarde de dezvoltare software

7.3.5. Standardul OSEK – prioritati

Schedulerul decide care este urmatorul task din taskurile ready ce sunt trasferate in starea running.

Reguli de prioritate:

1. Intreruperile au prioritate fata de taskuri
2. Nivelul de procesare al intreruperilor consta in unul sau mai multe nivele de prioritate
3. ISR au nivelele de prioritate asignate static.
4. Asignarea ISR la nivele de prioritate al intreruperilor este dependent de implementarea hardwarelui.
5. Pentru prioritizarea taskurilor si resurselor numarul mai mare pentru prioritati se refera la o prioritate mai mare.
6. Prioritatea taskurilor este asignata static de catre utilizator.



7.3. Standarde de dezvoltare software

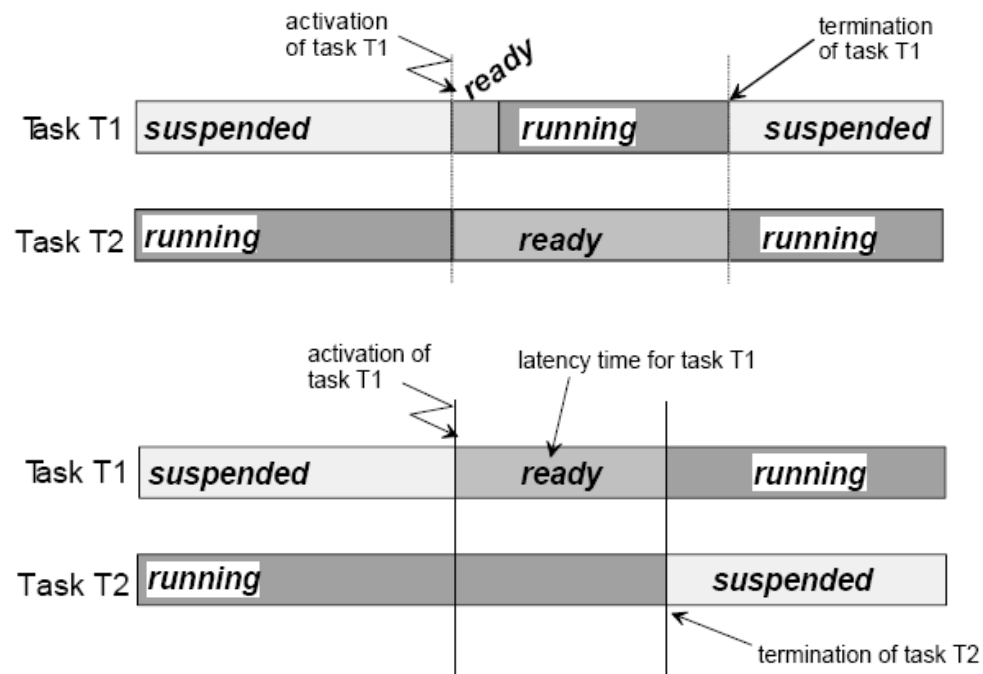
7.3.5. Standardul OSEK – prioritati

Exista trei tipuri de metode de planificare a taskurilor:

1. Full preemptive

2. Non Preemptive

3. Mixt



7.3. Standarde de dezvoltare software

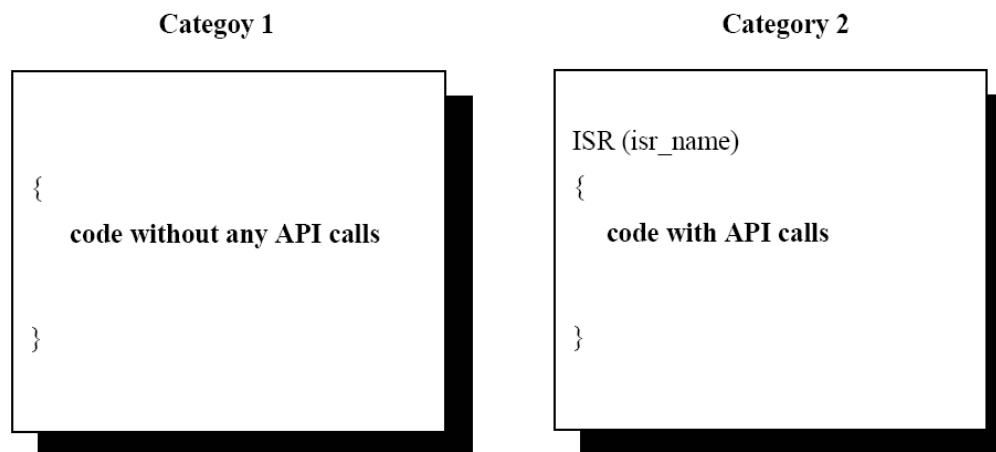
7.3.6. Standardul OSEK – intreruperi

OSEK defineste trei nivele de ISRs.

Level 1 - ISRs sunt executate imediat

Level 2 – ISR contine code ce apeleaza o functie a OS.

Level 3 – mode hybrid (coexista code ce nu apeleaza servicii OS cu cel ce apeleaza servicii OS)



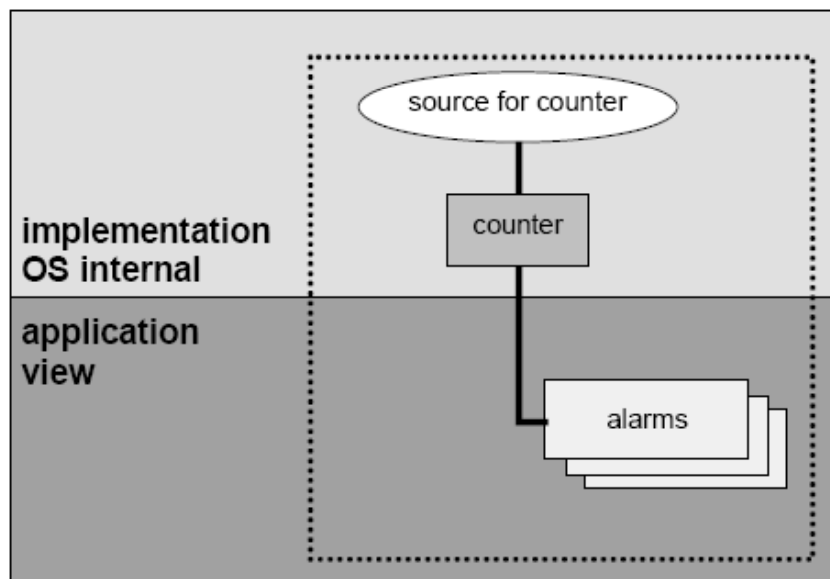
7.3. Standarde de dezvoltare software

7.3.7. Standardul OSEK – alarme

OSEK OS furnizea servicii de procesarea a evenimentelor repetitive. Exista doua tipuri de concepte ce proceseaza astfel de evenimente: countere si alarme.

Counterele se masoara in “ticks” si pot reprezenta timp, numere sau pulsuri receptionate. Nu exista standard API pentru manipularea lor. OSEK OS ofera cel putin un counter ce deriva dintr-un timer.

Alarma este este static asignata unui counter, task sau actiune. Exista dou tipuri de alarme: ciclice si single.



7.3. Standarde de dezvoltare software

7.3.8. Standardul OSEK – management

Resource management controleaza accesul la memorie, hard, etc. Resursele interne nu sunt vizibile utilizatorului putind fi accesate doar intr-o maniera specifica.

O resursa este automat “luata” de catre un task in momentul in care taskul trce in starea running (exceptie facand cazul cand resursa a fost deja asignata taskului in procesul de generare a sistemului). Ca si rezultat, prioritatea taskului este automat modificata. In momentul in care taskul isi tremina executia, resursa este eliberata..

Bibliografie

Bibliografie

- [1] IABG Information Tehnology, *"V –model. Lifecycle Process Model – Brief Description"*, February, 1993
- [2] <http://en.wikipedia.org/wiki/V-Model>
- [3] *"The V model – Relevant to Professional Scheme Paper 2.1"*, 2006
http://www.accaglobal.com/pubs/students/publications/student_accountant/archive/sa_0106_sskidmore.pdf
- [4] Jonathan Panell, *"V Testing Model"*, 2004
http://www.surfersjunkyard.com/Software_Testing_/Testing_Methodology/V_Testing_Model/v_testing_model.html
- [5] Claro Testing Inc., *"The Dangerous and Seductive V Model"*, Testing Experience Magazine, 2008
<http://www.clarotesting.com/page11.htm>
- [6] Bart Broekman, Edwin Notenboom *"Testing Embedded Software"*, Addison-Wesley, London, 2003
ISBN 0 321 15986 1