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: ***O****ffene* ***S****ysteme und deren Schnittstellen für die* ***E****lektronik in* ***K****raftfahrzeugen*
engleza: *Open systems and the Corresponding interfaces for Automotive electronics*
VDX - Vehicle Distributed eXecutive

Standardul cuprinde trei parti:

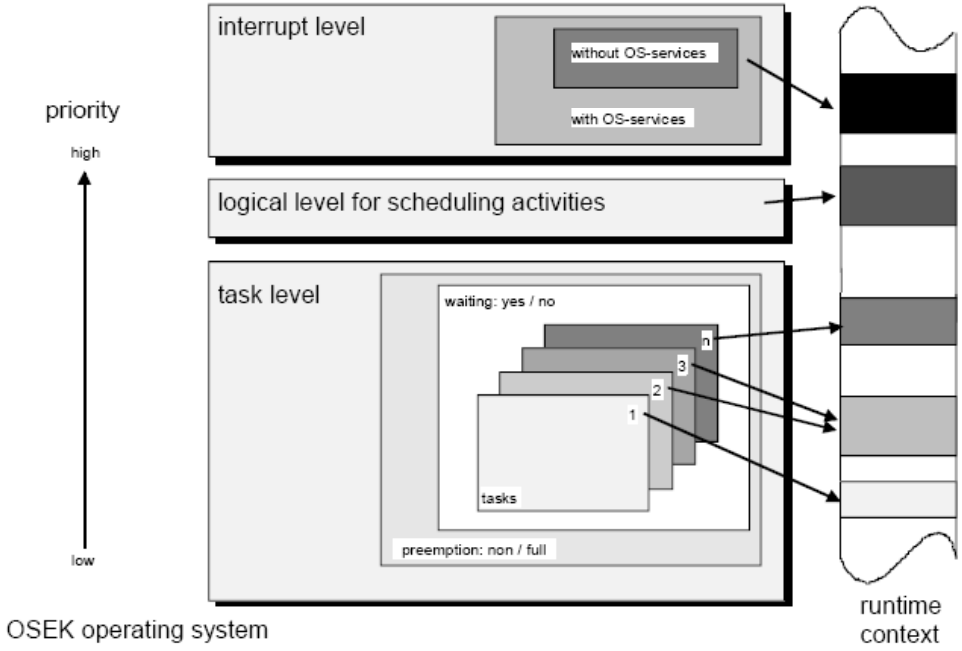
1. Operating System
2. Communication stack
3. Network Management

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

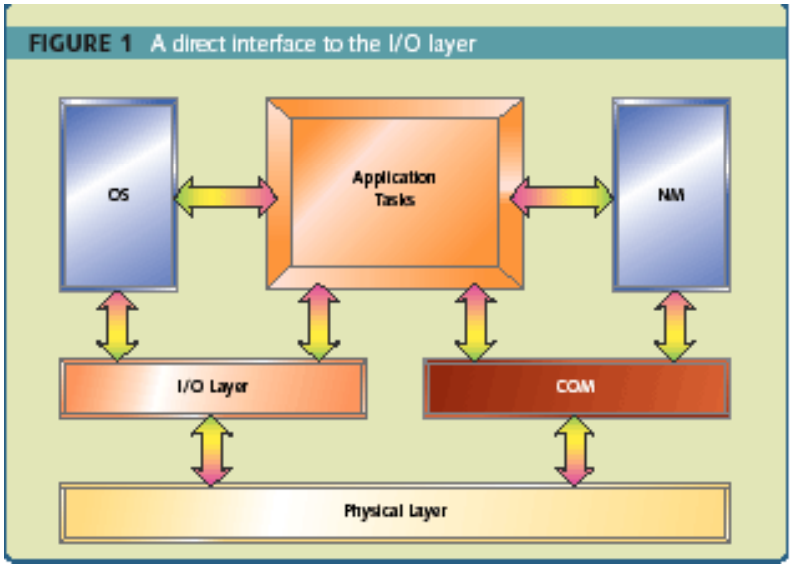
Sistemul de operare OSEK serveste ca si baza pentru aplicatii independente intre ele, creand un mediu ce ruleaza pe un procesor. OSEK OS permite controlul unui mediu de executie real-time unde diferite procese ruleaza in paralel. OSEK OS furnizeaza diferite interfete pentru user, aceste interfete fiind folosite de entitati ce “concureaza” pentru CPU. Exitsa dou tipuri de entitati: ISR (Interrupt Service Routines) si Taskuri.
OSEK defineste trei nivele de procesare: nivelul intrerupere, nivelul logic pentru scheduler si nivelul task.

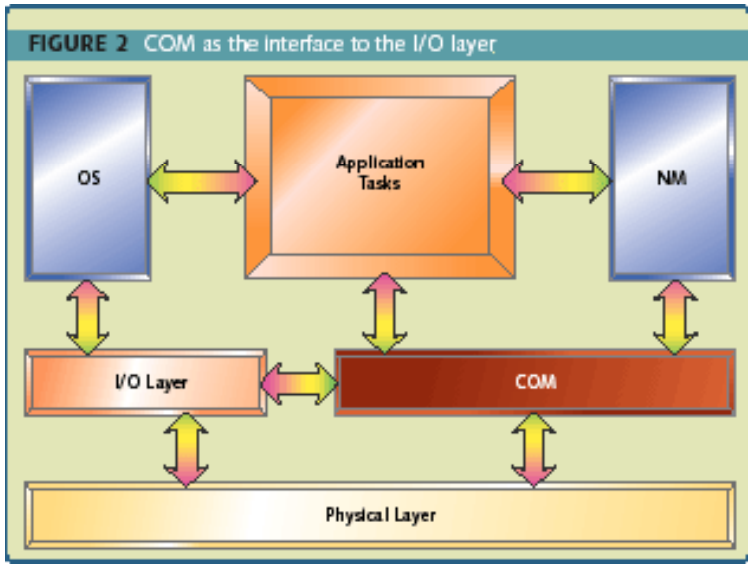


7.3. Standarde de dezvoltare software

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.

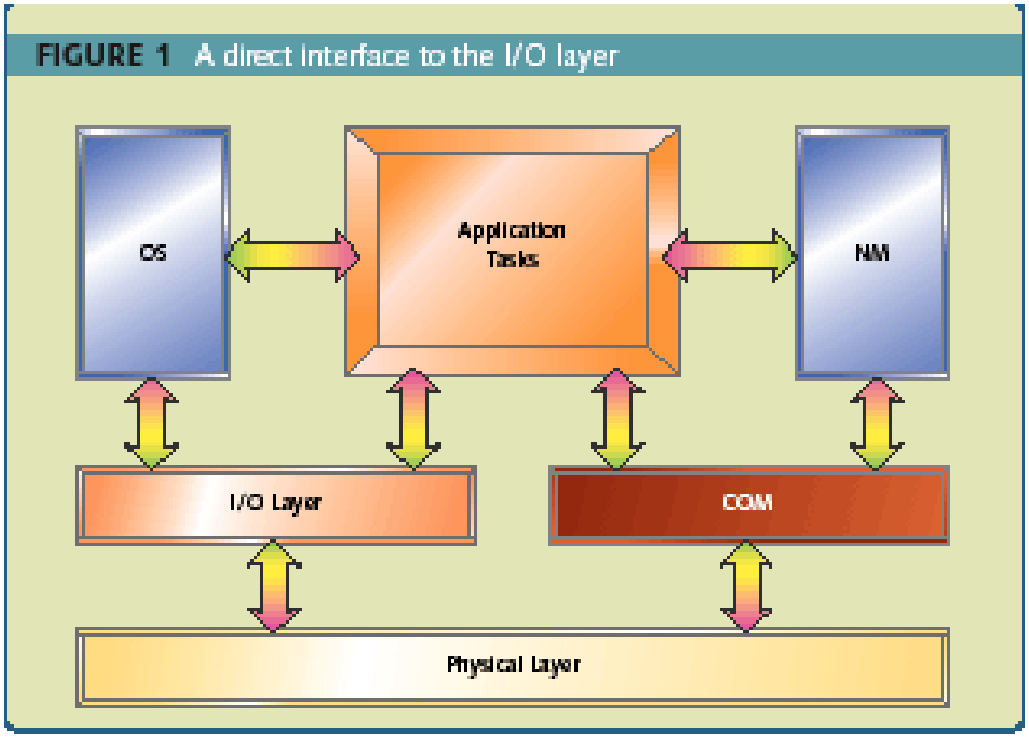




7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

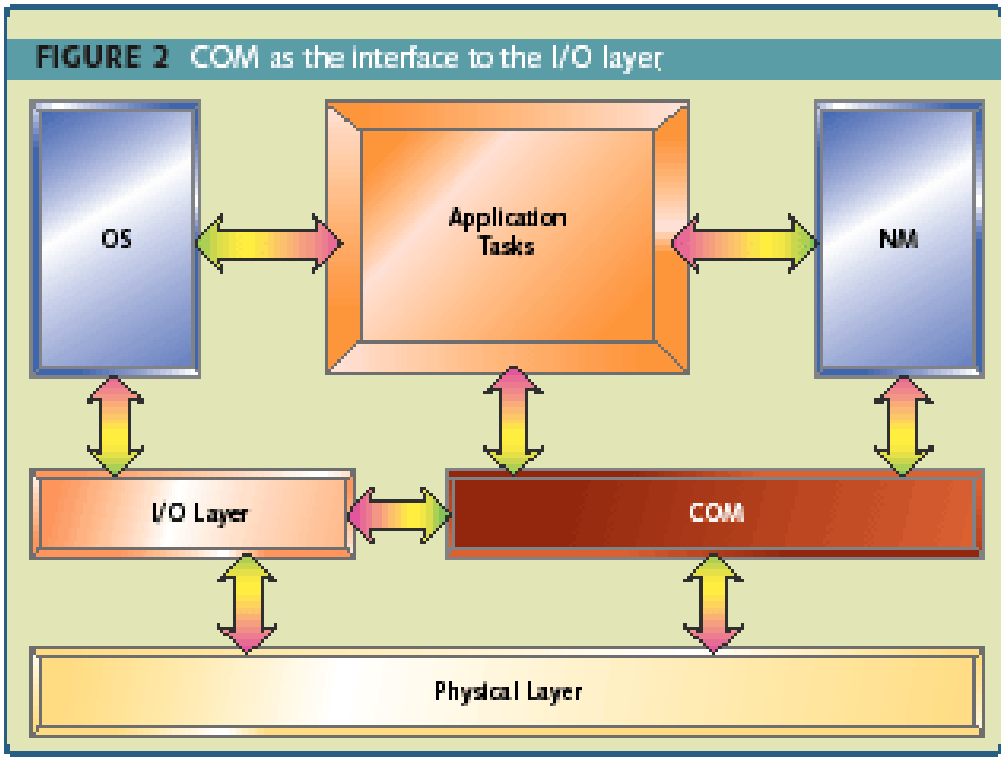
In prima forma, aplicatia adreseaza nivelul I/O direct. Aceasta parte nu exte definita in standardul OSEK datorita varietatii cerintelor diferitelor aplicatii.
Avantajul acestei forme de implementare este raspunsul rapid la o cerere de accesare a I/O din partea task-urilor.
Dezavantajul major este portabilitatea task-urilor.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

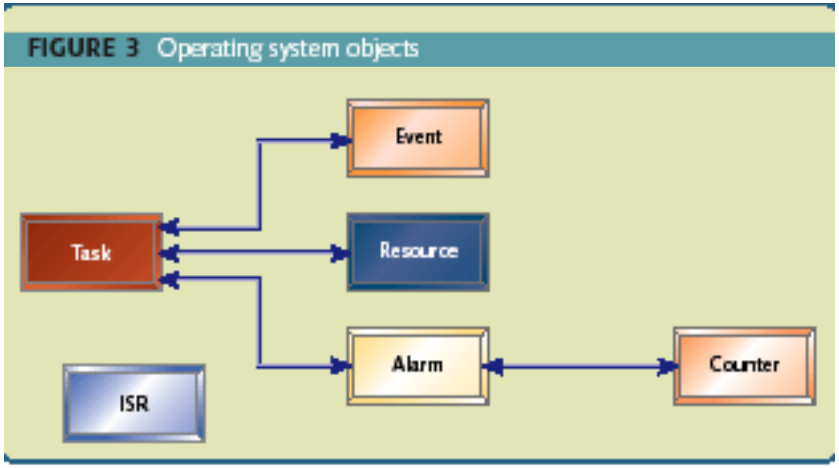
A doua forma trateaza nivelul I/O ca si un task. Fiecare aplicatie cere sau trimite informatii catre modulul COM.
Avantajul major este portabilitatea.
Dezavanatjul major consta in cresterea timpul de acces la I/O.



7.3. Standarde de dezvoltare software

7.3.3. Standardul OSEK – arhitectura

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



7.3. Standarde de dezvoltare software

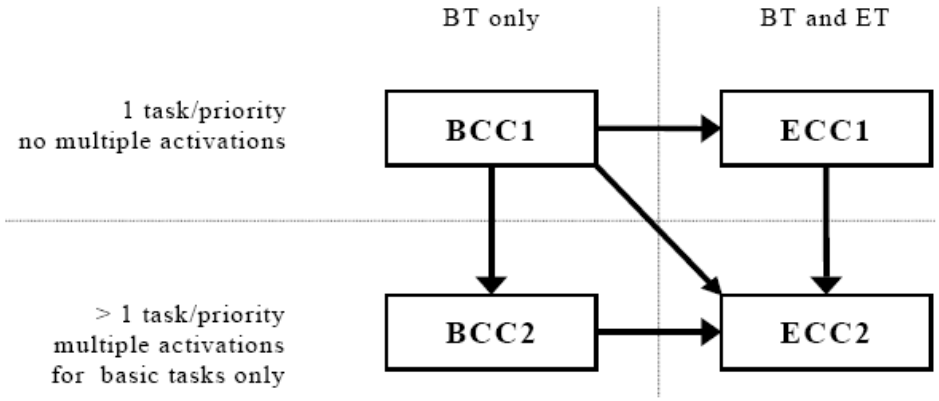
7.3.3. Standardul OSEK – arhitectura

Task-urile pot fi de mai multe tipuri: basic (BT) sau extended (ET), preemtive sau non-preemptive. Taskutile 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 preemtive pot fi intrerupte de un task de prioritate mai mare atunci cadn 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 clae de conformitate
Clasele de conformitate ne ajuta la implementarea partiala a standardului.
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

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

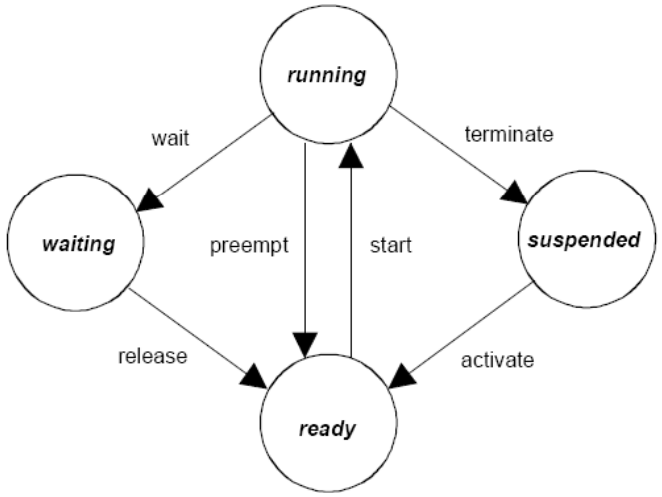
Portabilitatea aplicatiilor pote fi asumata daca minimul de cerinte sunt satisfacute. Cerintele minime pentru clasele de conformitate sunt:

	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 *suspended* 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.

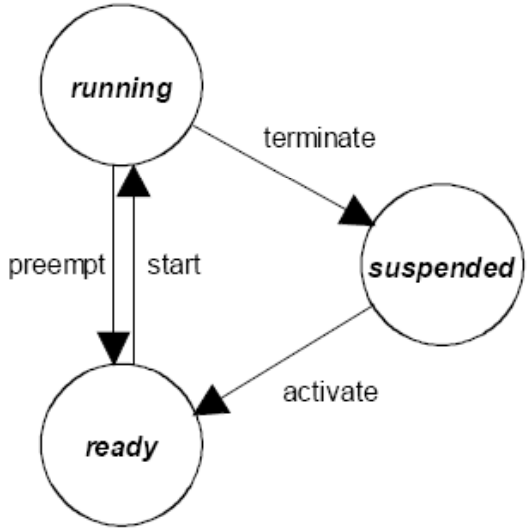


Transition	Former state	New state	Description
activate	*suspended*	*ready*	A new task is set into the *ready* state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	*ready*	*running*	A *ready* task selected by the scheduler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



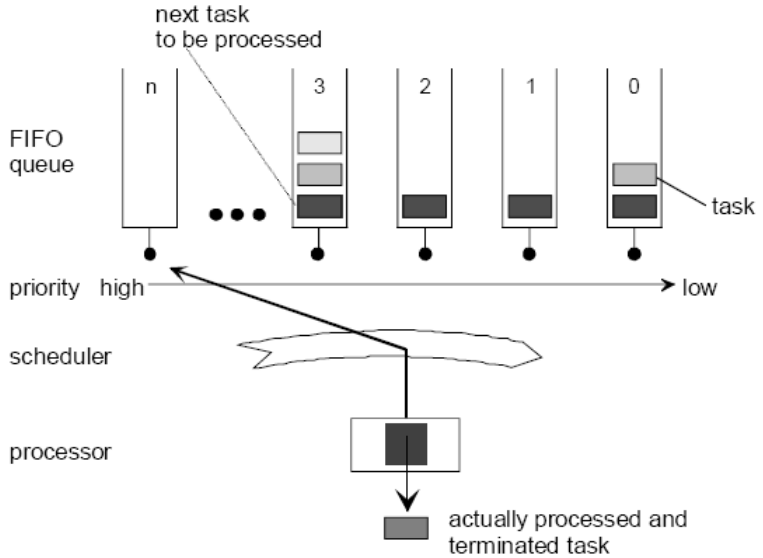
Transition	Former state	New state	Description
activate	*suspended*	*ready*[4]	A new task is set into the *ready* state by a system service. The OSEK operating system ensures that the execution of the task will start with the first instruction.
start	*ready*	*running*	A *ready* task selected by the scheduler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

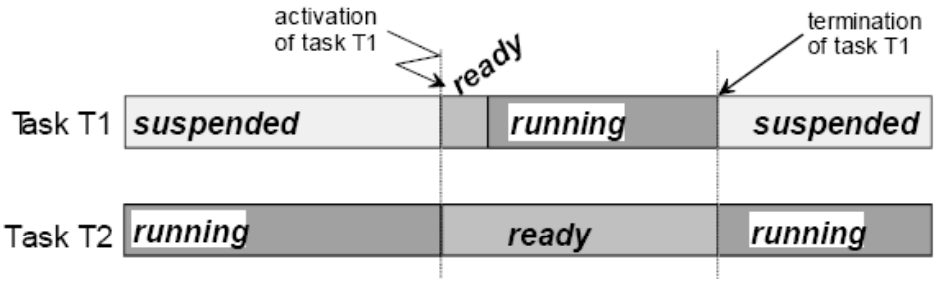


7.3. Standarde de dezvoltare software

7.3.5. Standardul OSEK – prioritati

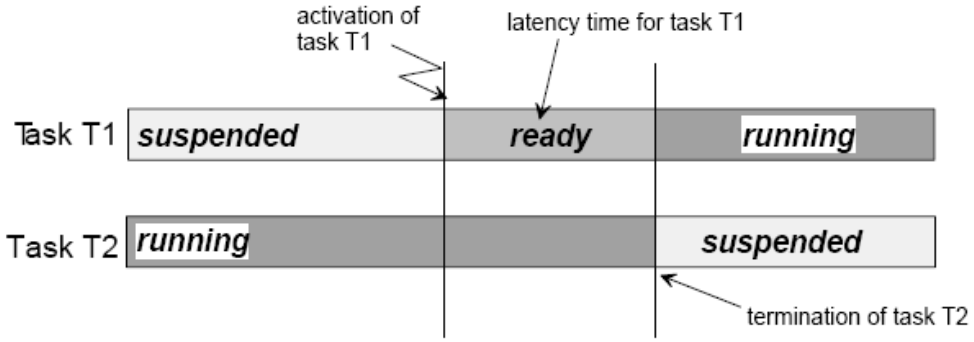
Exista trei tipuri de metode de planificare a taskurilor:

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)

Categoy 1

```
{
    code without any API calls
}
```

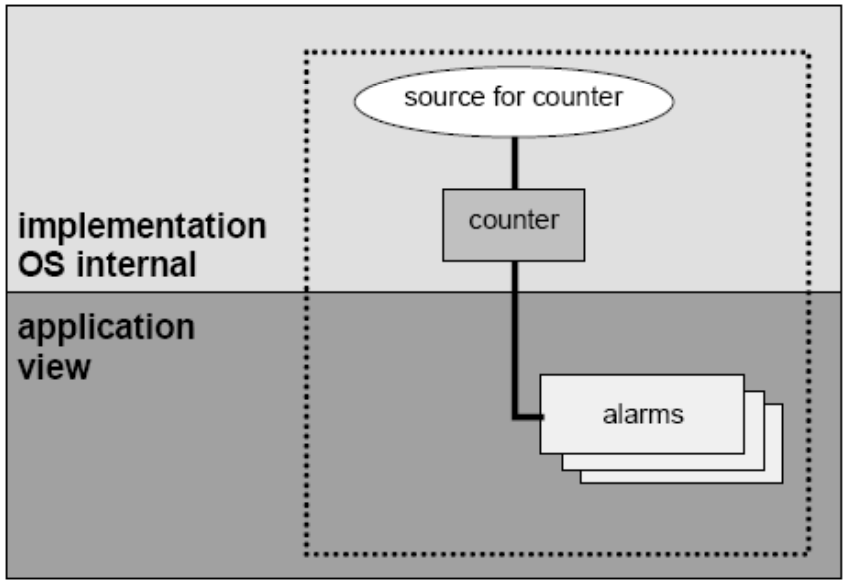
Category 2

```
ISR (isr_name)
{
    code with API calls
}
```

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