2. BÖLÜM

Project: Sample Problem 3.1
Analyst: SM
Simulation run ended at time: 480.0

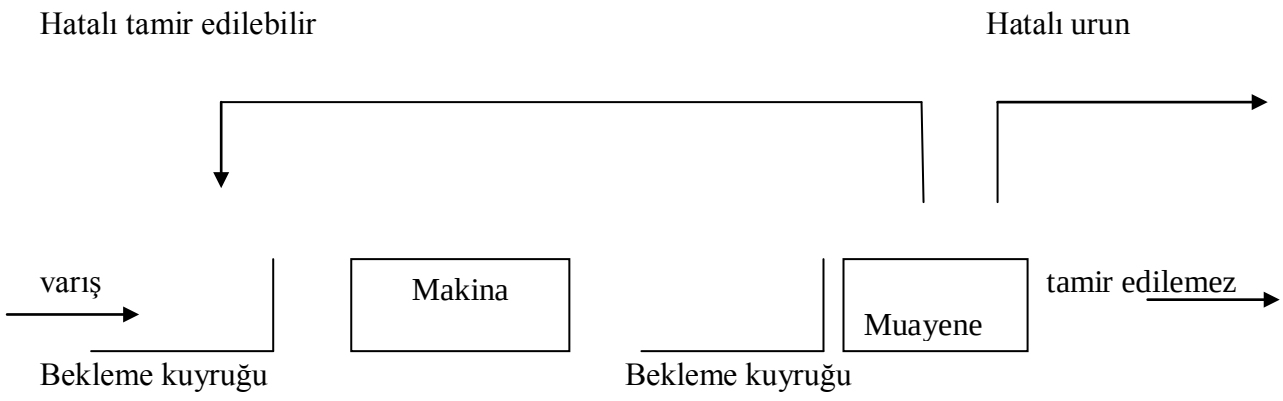
COUNTERS

Identifier	Count	Limit
JobsDone	93	Infinite

- Bu raporda 93 iş parçasının 480 dakikalık benzetim boyunca işlem gördüğü belirtilmektedir.
- Bu bir deneme sonucudur ve doğru sonucun bir tahminidir.

Sample Problem 3.2

İki iş türü ve 2 * iş istasyonu var.



Gelen işler bir makinadan işlem alabilmek için beklerler. Bu makinada işlemi tamamlanan iş 2. iş istasyonunda muayene

için bekler. Burada iyi, tamir edilebilir hatali veya kötü olanlar sınıflandırılır.

İyi olanlar sistemden çıkar. Tamir edilebilirler geri dönerek yeniden işlem almak üzere makina önünde beklerler ve tamiredilemez olanalar yeniden sisteme doner.

- Varışlararası zaman aralıkları ~ exp (9)
- İki türlü iş var. 0.30 1. tür
  0.70 2. tür

- Kuyrukta 1. tür işler 2. tür işe göre önceliklidir. Kuyruğun ön kısmında 1. Tür işler bekler. Bunlar kendi aralarında varış zamanlarına göre sıralanır. 1. tür işlerin arkasında 2. tür işler varış zamanl arına göre sıraya girerler.

- 1. tür iş geldiğinde, eğer 2. tür iş servis almakta ise işlem devam eder (kesilmez).

- Tamir edilebilir parçalar geri döndüğünde en düşük öncelikle sıraya girerler. Bu işler 1. Tür ve 2. Tür işlerin arkasında varış zamanlarına göre sıralanırlar.

- Aynı öncelik yeniden işlem gören parçalar için muayene istasyonunda da geçerlidir.

- Böylece makine ve muayene kuyrukları için üç öncelik tanımlanmıştır.

- 1. Tür iş, 2. Tür iş ve tamir edilebilir işler bu sırada önceliklidir.

- Her türlü iş için muayene zamanı ~TRIANGULAR (5., 8., 10.)

- Muayeneden sonra %80 iyi, %10 kötü, %10 tamir edilebilir. Bu yüzdeler her iki tür için geçerlidir.

- Makinenin setup ve işlem zamanını içeren toplam servis zamanı normal ve geri dönmüş işler için aşağıda normal dağılıma uygun olarak tanımlanmıştır.

	Normal iş		Geri dönmüş iş	
	Ort.	Std. Sapma	Ort.	Std. Sapma
1. Tür	5	2	3	1
2. Tür	4	1	2	1

- 480 dk için sistemin benzetimini yap
- İş türlerine göre tamamlanan iş sayılarını bul
- Reddedilen kötü çıktı sayısını bul

"ATTRIBUTE" ve "VARIABLES"

ATTRIBUTES Elementi

ATTRIBUTES: Number, Name (index), value, ... :
repeats;

(dizi tanımlanmışsa dizinin boyutunu temsil eder.)

VARIABLES: Number, Name (index), Value, ... :
repeats;

(değişken veya attribute bir değer vermek için kullanılan operand)

örnek problem için;

ATTRIBUTES: JobType: Status: Priority;

VARIABLE: StockLevel, 50; (StokLevel adlı değişkene 50 değeri verilir)

SIMAN'da 1 veya 2 boyutlu "array" tanımlamak mümkün. Format alt sınır ve üst sınırın tanımlanmasını içerir. Alt sınır tanımlanmamışsa DEFAULT değer 1'dir.

(Alt boyut sınırı .... Üst boyut sınırı) şeklinde verilir.

ARRAY(-5...5) bir boyutlu array. -5 ile +5 arasında.

Alt sınır 1, üst sınır 30 ise;

Vector(30) → bir boyutlu array , 1 ile 30 arasında değerler.

İki boyut için;

Matix(-3...6, 10) (*"," ile ayrılırsa iki boyutlu dizidir.*)

-3, 6 1-10

Ör: VARIABLES: 1, ReaderPoint:2-11, StockLevel(10);

Variable 1, adı *"ReaderPoint"*,
Varible2 , adı *"StockLevel"*

Reorder Point isimli array in ilk elemanı V(2)

StockLevel(1)

StockLevel array i 1 ila 10 arasında indis değerlerine sahip.

- Bu açıklık array içindeki eleman sayısına göre tanımlanır.

Ex: VARIABLES: 1, ReaderPoint, StockLevel(10);

{Burada değişken no.su "number operandi atlanınca" SIMAN tarafından otomatik verilir.}

- Tek boyutlu array'de üst ve alt limit arasındaki fark, üst ve alt sınır farkına eşit olmalıdır.

ARAY'E ELEMAN ATAMA

Bir dizinin elemanlarına başlangıç değeri vermek için value operandı kullanılır.

VARIABLES: Data(10,5);

"Data" adlı arrayın tüm elemanlarına (1 den 10' a) 5 değerini atar.
Örnek problemde "Mean" ve "Std" adlı değişken tanımlayarakmakinadaki normal dağılıma işlem zamanlarını tanımlayabiliriz.

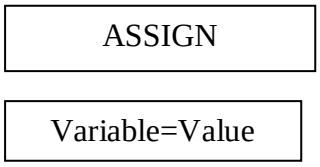
İki boyutlu tanımlarız. 1. İndex →İş türü {1 veya 2}
2. İndex →{regular or rework durumu (1 veya 2)} olarak tanımlanır.

VARIABLES: Mean(2, 2), 5,3,
4,2:

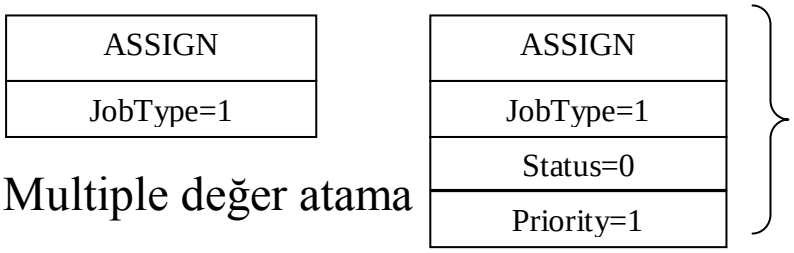
Std(2, 2), 2,1
1,1:

ASSIGN BLOCK İLE ATTRIBUTE VE VARIABLE'LARA DEĞER ATAMA

ASSIGN blok bir operation bloktur.



ASSIGN: Variable=Value; (Value, herhangi bir SIMAN expression'ı (deyimi) olarak kullanılabilir)



Multiple değer atama

Ex: ASSIGN:Jobtype=1; Status=0; Priority=1;
(Birden fazla değer verirken assignmentlar ":" ile ayrılır.)

KULLANICI TARAFINDAN TANIMLANAN KESİKLİ OLASILIK DAĞILIMINDAN ÖRNEKLEME (RASSAL DEĞİŞKEN ÜRETME)

2. örnek problemde iş türleri 0,30 → 1. Tür
0,70 → 2. Tür

Kesikli değerler 1 ve 2
İlişkili olasılıklar 0,30 1,0

DISCRETE(.3, 1, 1.,2) Bu Tanımdan rassal değişken 0,30 olasılıkla 1 0,70 olasılıkla 2 olarak doner.

Bundan elde edilen rassal sayının değerne göre iş türü 1 veya 2 olarak belirlenmiş olacaktır.

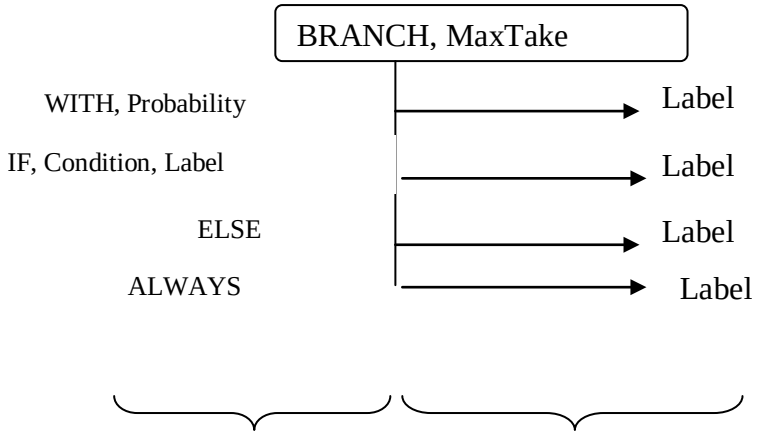
"JobType=DISCRETE(.3, 1, 1, 2)"

Bu kullanım SIMAN'da kullanıcı tarfından tanımlanan sürekli bir dağılımdan rassal değişken üretmeye de imkan verir. (Sürekli rassal değişken için birikimli olasılık fonksiyonu tanımlanır.)

ENTITY AKIŞINI BLOK İÇİNDE YÖNLENDİRME

Örnek problemde sistemden çıkmakta olan ürün (entiy) iyi, kötü veya reddedilen bir entity olması dikkate alınarak farklı yönlere gönderilme ihtiyacındadır.

Bu amaçla SIMAN'da BRANCH blok tanımlanmıştır. Bu blok tek amaçlı bir blok'tur.



BRANCH koşulu
BRANCH destination Label

MaxTake → Entitinin seçebileceği max. Branch sayısı (integer)

Seçilebilecek BRANCH sayısı ∅ → MaxTake değişir.

Hiçbir BRANCH seçilmezse entity → dispose olur. (modelden çıkar)

Yalnızca Bir BRANCH seçildiğinde entity seçilen BRANCH yönünde hareketine devam eder.

Birden fazla BRANCH seçildiğinde, ilk seçilen BRANCH yönünde hareket eder. Entitinin aynı attributes değerine sahip kopyası yaratılarak diğer seçilen BRANCH'lara gönderilir.

Orjinan entity Primal entity, kopya olarak yaratılan Entity ise Secondary entity olarak adlandırılır.

BRANCH Koşulları:

1) OLASILIKLI BRANCH; WITH, P $0 \leq P \leq 1$

ex: WITH, .8 , bu branch'ın 0,8 olasılığıyla seçileceğini gösterir.

2) KOŞULLU BRANCH

Variable ve attribute'ları içeren mantıksal koşula dayalı seçim.

IF, C C: herhngi bir geçerli deyim.
IF, ArrivalTime < TNOW ; doğru ise bu branch seçilir.

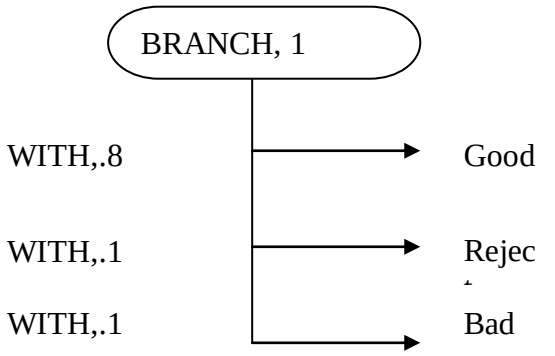
3) DETERMINISTIC BRANCH

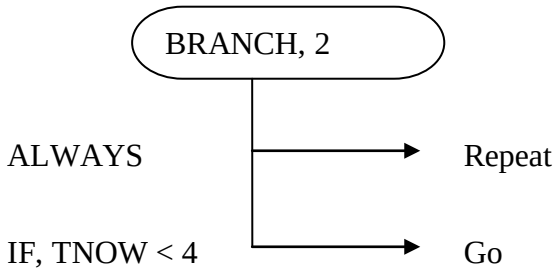
Null condition,

ALWAYS or ELSE kullanılarak deterministic branch tanımlanır.

ALWAYS ise her varış için bu bloğa →
ELSE ise maxTake sayısı kadar önceki BRANCH'ları alınmamışsa, bu branch seçilir.

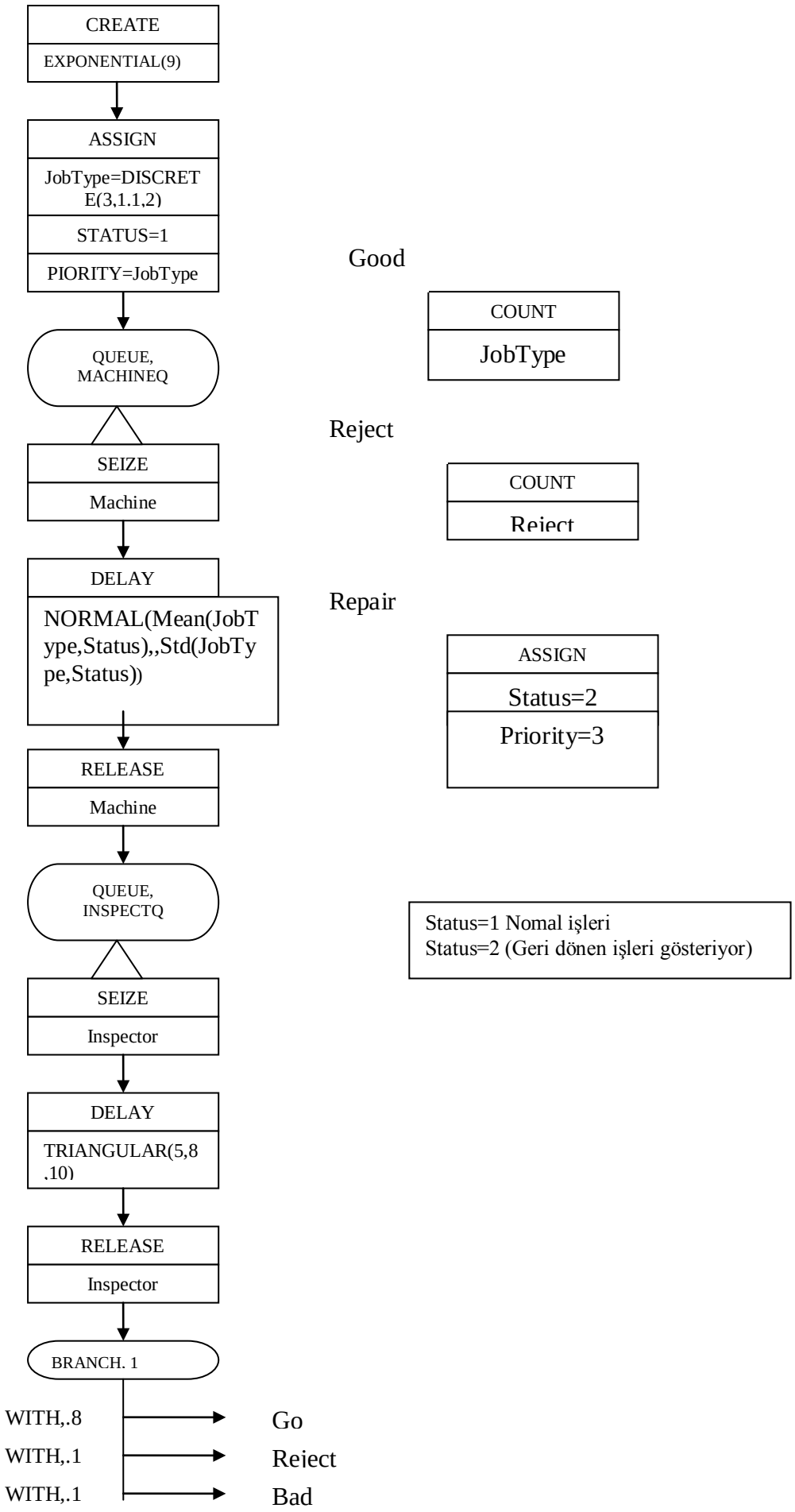
ornek:





Burada her entity daima Repeat Labelına gider. Buna ilaveten eğer TNOW'un o anki değeri < 4 ise, gelen entitinin bir kopyası "Go" Labellı bloğa gider.

Şimdi Örnek bir problem çözebiliriz.


CREATE
EXPONENTIAL(9)
ASSIGN
JobType=DISCRETE(3,1.1,2)
STATUS=1
PIORITY=JobType
QUEUE, MACHINEQ
SEIZE
Machine
DELAY
NORMAL(Mean(JobType,Status),,Std(JobType,Status))
RELEASE
Machine
QUEUE, INSPECTQ
SEIZE
Inspector
DELAY
TRIANGULAR(5,8,10)
RELEASE
Inspector
BRANCH. 1
WITH,.8
WITH,.1
WITH,.1
Go
Reject
Bad
Good
COUNT
JobType
Reject
COUNT
Reject
Repair
ASSIGN
Status=2
Priority=3
Status=1 Nomal işleri
Status=2 (Geri dönen işleri gösteriyor)

```
BEGIN;
        CREATE: EXPONENTIAL (9);            sisteme gir
        ASSIGN: JobType=DISCRETE(.3,1,1.,2):
        JobType attribute
                Status=1:                   Status attribute
                Priority=Jobtype;               Priority
attribute
MERGE  QUEUE,  MACHINEQ;            Makine için bekle
        SEIZE:     Machine;                 makineya gir
        DELAY: NORMAL(Mean(JobType, Status),
                        Std (JobType, Status));
        RELEASE: Machine;
        makineyı boşalt
        QUEUE, InspectQ;            Inspector için bekle
        SEIZE: Inspector;                   Inspectora gir
        DELAY: TRIANGULAR(5,8,10);          İşlem zamanı
        RELEASE: Inspector;                     boşalt
        BRANCH, 1:

                WITH, .8, Good:             iyi parça
                WITH, .1, Reject:           red parça
                WITH, .1, Bad;              kötü parça

Good COUNT:     JobType: DISPOSE;  iyi parça iş türüne göre sayiliyor
Reject COUNT:   Rejects: DISPOSE;       Red edileni say
Bad ASSIGN: S   Status=2:               Status att. Reset et
                Priority=3: NEXT(Merge);

                                    reddedilen parçayı en düşük
                                    öncelikle MachineQ y gönder.
```

```
END;


BEGIN;
PROJECT, Sample Problem 3.2, SM;
ATTRIBUTES:      JobType:
                 Status:
                 Priority;
VARIABLES:       Mean(2,2),     5,3,
                                4,2:
                 Std(2,2),  2,1,
                            1,1;

QUEUES:          MachineQ,      LVF(Priority):
                 InspectQ,      LVF(Priority);

RESOURCES:       Machine:
                 Inspector;

COUNTERS:        Type1 Jobs Done:
                 Type2 Jobs Done:
                 Rejects;
REPLICATE,       1,0,480;
END;
```

Project: Sample Problem 3.2
Anlayst: SM
Simulation run ended at time: 480.0

	COUNTERS	
Identifier	Count	Limit
Type 1 JobsDone	23	Infinite
Type2 JobsDone	24	Infinite
Rejects	7	Infinite

TIME DEPENDANT DATA KAYITI

Bir veya birden fazla zamana bağlı değişkenin istatistiğinin tutulmasında kullanılır.

DSTATS: Number, Expression, Name, OutFile:
Repeats;

Number: DSTATS sayısı, 1'den,…..2,3,4 etc.
Experiment processor bu sayıyı
otomatik atar.,

Genellikle;

NQ(QueueID) = Kuyruktaki entiti sayısı
NR(ResourceID)=Meşgul kaynak sayısı
Kullanılır.

Name: Number ile birlikte dstat entry tanımında kullanılır.
Summary reporta dstat değişkeni istatistiği
Olarak dönecek özel amaçlı SIMAN değişkeninin argumanı olarak
Kullanılır. Aynı zamanda summary reportta gösterilen dstat variable ın summary istatistik label'ı olarak kullanılır.

Out File: Output file

Ex: DSTATS: NQ(Buffer), BufferSize:
NR(Machine), MachineUtil., "C:\Machine.dat";

İki adet DSTAT değişkeni tanımlar. 1. Buffer Size, Buffer adlı kuyruk istatistiğini tutar, 2.si MachineUtil, Machine adlı resource'da meşgul brim sayısını tutar ve datayı C:\Machine.dat file'na save eder.

GÖZLEMLENEN DATA KAYDI

TALLY: TallyID, Value;

Bir bloğa giren entitinin bir değerini kaydeder.

TALLY: TallyID, Value;

TallyID: Gözlemin kaydolacağı sayı veya isim
Value: Hangi Değer kaydolacağı

TALLY
TallyID,Value

TALLY
1, BatchSize

TALLY
TBDeparte, BET

BatchSize 1 nolu TALLY'e kaydedilir.

TALLY
PaitType, INT(Time)

Bloğa varışlar arası zaman aralığı TBDepaitsa kaydedilir.

BET: Time between succesive arrivals

Time adlı attribute markalanmış zaman kullanılarak bulunan
zaman aralığı, PaitType isimli attribute içinde ihtiva edilen sayı ile tanımlı yere kaydedilir. Bu durumda bir TALLY block registersların sembolik adı yerine attributesların nümerik değerini kullanarak birçok registers'a karşılık gelen değerleri kaydeder.

TALLIES Elementi
Modelde Bir veya Birden fazla Tally Kullanılırsa

TALLIES: Number, Name, OutFile:
Repeats;

Ex:
TALLIES: TSysOne:
TSys Two, "C:\TSysTwo.dat";

2 register tanımlar. 1. TSys One
2. TSys Two

SAMPLE PROBLEM 3.3

Örnek 3.2ye ilave olarak

a) Kuyruk uzunlukları
b) Kaynak Kullanım durumları
c) İş parçalarının sistemde geçirdiği zaman
d) Inspector'un iş türüne göre kullanım durumu istatistiklerini bulacak ek SIMAN segmentları:

SIMAN'da bir entitinin 2 nokta arasında geçirdiği zamanı kaydetmek için MARK modifier'ı model içinde herhngi bir blokta kullanılabilir. Bu modifier aktif entitinin bu bloğa varış zamanını kaydeder.

MARK (AttributeID)

Mark time kaydı için kullanılan attribute tanımlayıcı

Entiti, modeldeki bu 2 noktayı da geçince, zamanı kaydetmek için yine TALLY blok kullanılır.

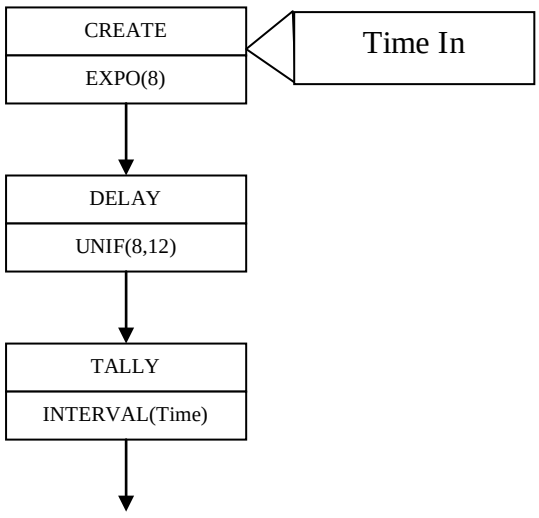
INTERVAL(AttributeID)

Mark time da kulanılan att. Tanımlayıcı

Genellikle;

INT( ) olarak kulanılır.

TALLY blok entitinin modelden çıktığı noktada kullanılır.

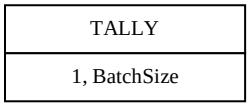


CREATE bloğa giren entity'nin giriş zamanı TimeIn isimli attribute içinde MARK tarafından kaydedilir.

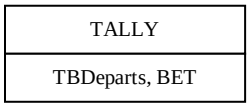
- Bu, TimeIn=TNOW işlemine eşdeğerdir.
- TALLY blok'da INTERVAL(TimeIn)

TNOW – TimeIN = zamanı kaydedilir. Böylece entitinin CREATE blok ile TALLY blok arasında geçirdiği zaman bulunur.

TALLY bloğun diğer KULLANIMLARI

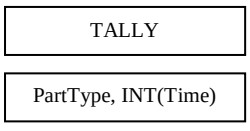


BatchSize attribute değeri 1 nolu TALLY içinde kaydedilir.



Bu bloğa gelen varışlar arasındaki zaman TBDeparts içinde kaydedilir

- BETWEEN or BET ile SIMAN varışlar arası zaman aralığı kaydı yapar.



PartType adlı attribute içindeki numara ile tanımlı kaydedici içinde kaydedilir. Bu durumda bir TALLY blok birçok kaydediciye kayıt yapar.

TALLY: TallyID, Value;

TallyID: Gözlemler için Kaydedici (Number or name)

Value: Her entity varışında kaydedilecek değer

(Model Framede)

TALLY
TallyID, Value

EXP FILE İÇİN TALLY KAYITLARI

TALLIES: Number, Name, OutFile:
Repeats;

Number ve Name TallyID'de kullanılabilir.

OutFile: çıktı file.
Optional'dır.

TALLIES: TSysOne:
TSysTwo, "C:\TSysTwo.dat";
(TSysTwo ile gözlenen gözlemler bu file da saklanır.

STORAGES: Number, Name:
repeats;
*****

Örnek:

STORAGES: InspType1:
InspType2;

NSTO(InspType1) veya NSTO(1)

NSTO(InspType2) veya NSTO(2)

- Bu storage'ların içindeki (o anki) entity sayısını verir.
- Bu örnekte Number operandı atlandı.
- SIMAN prosesörü bu storage'lara otomatik olarak 1 ve 2 numaralarını verir.
- Örneğimizde, STORAGES elementini inspector isimli kaynağın iş türlerine göre doluluk oranını ayrı ayrı kaydetmek için kullanıyoruz.

BEGIN;
CREATE: EXPO(9);
Mark(TimeIn); *{Entity'nin bu bloğa giriş zamanı}*
ASSIGN: JobType=DISCRETE(.3,1,1.,2)
Status=1:
Priority=JobType;
Merge QUEUE, MachineQ;
SEIZE: Machine;
DELAY: NORMAL(Mean(JobType, Status), Std(JobType, Status))
RELEASE: Machine;
QUEUE, InspectQ,
SEIZE: Inspector;
DELAY: TRIA(5, 8, 10), Jobtype;
RELEASE: Inspector;
BRANCH, 1: *dağılımdan üretilen değersaklanır.*
WITH, .8, Good: *JobType=1 veya 2'dir. Bunun için 2* WITH, .1, Reject: *starage tanımlandı.}*
WITH, .1, Bad;
Good TALLY: JobType, INT(TimeIn): DISPOSE;

Reject COUNT: Rejects: DISPOSE; *{count the rejects}*
Bad ASSIGN: Status=2:
Priority=3: NEXT(Merge)

Not: Eğer bir storage facility identifier DELAY blokta belirtilirse, bir entity DELAY bloğa geldiğinde SIMAN otomatik olarak NSTO(StorID) değerini artırır. Çıktığında da azaltır.

PROJECT, SampleProblem 3.3, SM;
ATTRIBUTES: TimeIn:
JobType:
Status:
Priority;
VARIABLES: Mean(2, 2), 5, 3,
4,2:
Std(2, 2), 2, 1,
1, 1;
QUEUES: MachineQ, LVF(Priority):
InspectQ, LVF(Priority);
RESOURCES: Machine:
Inspector;
STORAGES: InspType1: *{Modelde 2 storage tanımlanır. InspType1, InspType2* InspType1; *aynı zamanda 1, 2 dir. DELAY'de tanımlı JobType da 1 ya da 2 değeri alır. Buna göre burada sayaç olur.*
COUNTERS: Rejects;
TALLIES: Type1 Time in Sys: *{JobType (1 veya 2 için istatistik tutar)}*
Type2 Time in Sys;
DSTATS: NQ(MachineQ), *{Kaynak uzunlukları, kaynak kullanımı ve storage kullanımı istatistikleri}*
NQ(InspectQ),
NR(Machine),

```
                NR(Inspector),
                NSTO(InspType1),
                NSTO(InspType2),
REPLICATE, 1, 0, 480;
END;
```

p.115'deki report dosyasini inceleyin.

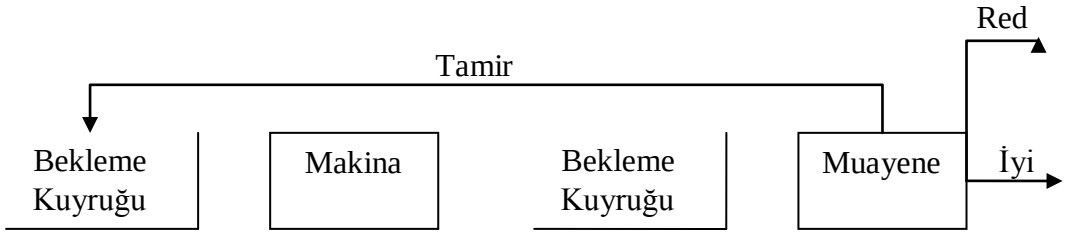
- NR(Inspector), overall utilization'ı verir. Fakat iş türlerine göre vermez. Bu bilgi için, inspection delay'i veren DELAY blok operand olarak her iki iş türü için ayrı storage tanımlanır. Böylece iş türüne göre kullanım oranı, iş sayısına göre istatistikleri iki storage içinde kaydedilir. STORAGE'lar DELAY blokla ilişkilendirilir ve delay içindeki entity sayısı'nın sayacını storage'lar sağlar.

Problem 3.3.

Örnek 3.3.'de tamir edilebilen işlerin machine bloğa döndüğünde kendilerine ait ayrı bir kuyruğu olsun (LVF ile kendi içlerinde sıralansın). İşler normal gelen işlere göre öncelikli olsun. Tamir edilebilen iş kuyruğu için istatistikleri kaydetsin.

```
BEGIN;
    CREATE:     EXPO(9);
                Mark(TimeIn);
    ASSIGN:     JobType=DISCRETE(.3,1,1.,2)
```

```
                Status=1:
                Priority=JobType;
Merge           QUEUE, MachineQ;
        SEIZE, 2: Machine;
MachDel DELAY:      NORMAL(Mean(JobType,      Status),
Std(JobType, Status))
        RELEASE: Machine;
        QUEUE, InspectQ,
        SEIZE: Inspector;
        DELAY: TRIA(5, 8, 10), Jobtype;
        RELEASE: Inspector;
        BRANCH, 1:
            WITH, .8, Good:
            WITH, .1, Reject:
            WITH, .1, Bad;
Good            TALLY: JobType, INT(TimeIn): DISPOSE;
Reject          COUNT: Rejects: DISPOSE;
Bad     ASSIGN: Status=2:
                    Priority=3;
        QUEUE, ReworkQ;
        SEIZE, 1: Machine: NEXT(MachDel); {Machine
kullanmada öncelik}
END;
```




```
PROJECT,            SampleProblem 3.3, SM;
ATTRIBUTES:     TimeIn:   JobType:
                Status:         Priority;
```

```
VARIABLES:      Mean(2, 2),     5, 3,
                                4,2:
                Std(2, 2), 2, 1,
                                1, 1;
QUEUES:         MachineQ: InspectQ,     LVF(Priority):
                ReworkQ,        LVF(Priority);
RESOURCES:      Machine:
                Inspector;
TALLIES:        Type1 Time in Sys:
                Type2 Time in Sys;
DSTATS:         NQ(MachineQ), MachineQueue;
                NQ(ReworkQ), RejectsQueue:
                NR(Machine), Machine Util.:
                NR(Inspector), Inspector Util.;
COUNTERS:       Rejects;

REPLICATE, 1, 0, 480;
END;
```

(Sum. Rep. in 3)