

## Il Metodo ASM

## Metodo ASM (1)

- Il metodo basato sulle Abstract State Machine (ASM) permette lo sviluppo
  - **rigoroso** e **formale**
  - di sistemi software, ad elaborazione
    - sequenziale
    - parallela
    - distribuita

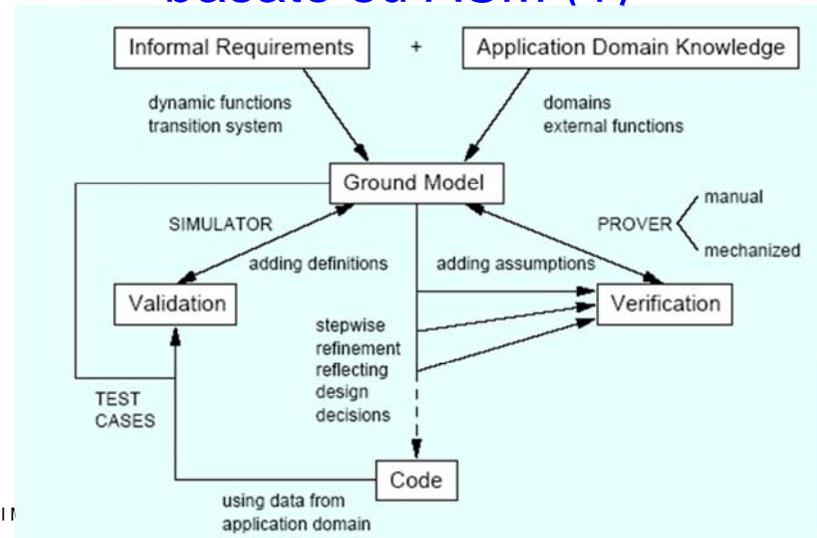
## Metodo ASM (2)

- È un metodo **pratico** di progettazione:
  - Formalizzare i requisiti in un ground model
  - Implementare il ground model con raffinamenti successivi, ottenendo una **gerarchia di modelli intermedi** (strutturazione verticale)
  - Strutturare e costruire il sistema **orizzontalmente**

## Obiettivo

- Definire un framework concettuale **semplice** e **preciso** per supportare e integrare
  - le principali attività di sviluppo sw
  - le principali tecniche di modellizzazione e analisi

## Modelli e Metodi nello sviluppo basato su ASM (1)



Il

## Modelli e Metodi nello sviluppo basato su ASM (2)

- Supporto per le attività di
  - Definizione dei requisiti: costruzione del Ground Model (GM)
    - Descrizione rigorosa e concisa del sistema, espressa secondo termini specifici del dominio applicativo, per mezzo di un linguaggio comprensibile da tutte le parti coinvolte
  - Architettura e progetto delle componenti
  - Validazione dei modelli, mediante simulazione
  - Verifica delle proprietà del modello (dimostrazioni)

Il Metodo ASM

Documentazione

6

## Modelli e Metodi nello sviluppo basato su ASM (3)

- Modellizzazione e analisi sulla base della sistematica separazione degli interessi:
  - Integrando descrizioni
    - Dinamiche (**operazionali**)
    - Statiche (**dichiarative**)
  - Combinando ad ogni livello di dettaglio voluto metodi di:
    - Validazione (simulazione)
    - Verifica (**dimostrazione**)

Il Metodo ASM

7

## Ground Model: Scopo (1)

- Descrivere i requisiti del sistema in modo evolutivo, in modo
  - consistente e non ambiguo
  - semplice e conciso
  - astratto e completo
- Il ground model deve risultare **comprensibile** e **verificabile** sia per gli specialisti del dominio applicativo che del dominio tecnologico

Il Metodo ASM

8

## Ground Model: Scopo (2)

- Comprensibilità
  - Implica una condivisione del linguaggio con cui il modello è espresso
  - ASM permette di calibrare il livello
- Verificabilità
  - Completezza e consistenza del modello

## Ground Model: Caratteristiche (1)

- Permette di rendere noto all'inizio dello sviluppo ciò che il sistema deve fare in forma di definizione matematica
- Tutti gli oggetti del GM (predicati, funzioni, trasformazioni) corrispondono a entità del mondo reale (proprietà, relazioni, processi)
  - Corrispondenza desiderata 1:1

## Ground Model: Caratteristiche (2)

- Il GM deve essere **preciso** rispetto al livello di astrazione prescelto e **flessibile**, in modo da poter essere facilmente modificato ed esteso
  - riusabilità
  - adattabile a diversi domini applicativi

## Ground Model: Caratteristiche (3)

- Il GM deve essere **semplice** e **conciso** per soddisfare la comprensibilità sia da parte dei progettisti che dei conoscitori del dominio applicativo

## Ground Model: Caratteristiche (4)

- Il GM deve essere sufficientemente **astratto** ma **completo**
  - ogni caratteristica semanticamente rilevante deve essere presente

## Ground Model: Caratteristiche (5)

- Il GM deve essere **validabile**

## Livello di astrazione opportuno

- È il problema principale della definizione del GM
- Il metodo ASM affronta il problema riducendolo a un problema di scelta del linguaggio opportuno per la comunicazione tra dominio applicativo e tecnologico

## Il problema del linguaggio

- Il metodo ASM affronta il problema usando solo notazioni per la descrizione dei fenomeni nel mondo reale che fanno riferimento a concetti algoritmici

## Creazione del Ground Model (1)

- Per formulare adeguatamente il GM è necessario rispondere alle seguenti domande
  - Quali sono gli **agenti** del sistema e quali relazioni intercorrono tra di essi?
    - In particolare, che relazione sussiste tra il **sistema** e il suo **ambiente**?
  - Quali sono gli **stati** del sistema?
    - Quali sono i **domini** degli oggetti e quali sono le funzioni, predicati e relazioni definiti su di essi? (approccio o.o.)
    - Quali sono le parti **statiche** e quali quelle **dinamiche** (inclusi input e output) degli stati?

## Creazione del Ground Model (2)

- Quali sono gli stati del sistema coinvolti dalle **transizioni**?
  - Sotto quali **condizioni** per gli agenti si verificano le transizioni?
  - Quali **effetti** sugli agenti hanno le transizioni?
  - Che cosa si suppone accada quando le condizioni **non sono soddisfatte**?
  - Quali forme di **uso erraneo** devono essere previste e quali meccanismi di gestione delle eccezioni devono essere implementati?
  - Quali sono le caratteristiche di **robustezza** desiderate?
  - Come sono collegate le **azioni interne** agli agenti alle **azioni esterne**?

## Creazione del Ground Model (3)

- Chi **inizializza** il sistema e in cosa consiste l'inizializzazione?
  - Che relazione esiste tra l'inizializzazione e l'input?
- Esistono condizioni di terminazione?
  - Se sì, come sono determinate?
  - Che relazione esiste tra la terminazione e l'output?
- La descrizione del sistema è **completa** e **consistente**?
- Quali sono le **assunzioni** relativamente al sistema e quali sono le **proprietà desiderate**?

## Esempio: Problema

- Modellizzare il comportamento di un sistema costituito da n ascensori che si muovono lungo m piani

## Esempio: Requisiti (1)

- Requisiti espressi in linguaggio naturale:
  - **REQ1**: Ogni ascensore è provvisto di una pulsantiera
    - ogni tasto corrisponde ad un piano
    - premendone uno, il tasto si illumina e l'ascensore si muove verso il piano corrispondente
    - quando l'ascensore raggiunge il piano selezionato si ferma e l'illuminazione del tasto si spegne

## Esempio: Requisiti (2)

- **REQ2**: Ogni piano ad eccezione degli estremi è provvisto di due tasti, per prenotare in salita e in discesa
  - premendone uno, il tasto si illumina
  - quando l'ascensore raggiunge il piano si ferma e l'illuminazione del tasto si spegne solo se
    - è diretto nella stessa direzione della prenotazione,
    - oppure se non ci sono prenotazioni pendenti
  - nell'ultimo caso, se sono premuti entrambi i tasti del piano, solo uno di questi perderà l'illuminazione

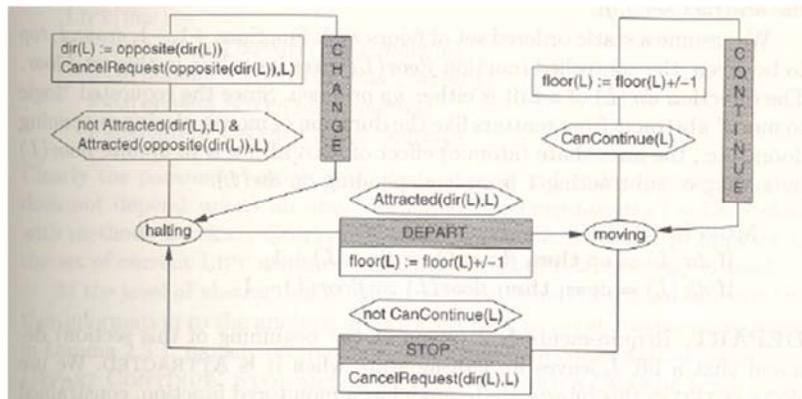
## Esempio: Requisiti (3)

- **REQ3**: Se non ci sono prenotazioni, l'ascensore rimane fermo al piano in cui si trova, in attesa che arrivino nuove prenotazioni
- **REQ4**: Sulla pulsantiera c'è anche un tasto che, se premuto,
  - invia un segnale di allarme
  - pone l'ascensore "fuori servizio"
  - il meccanismo per riportare "in servizio" l'ascensore è esterno al sistema

## Esempio: Osservazioni

- Gli ascensori si **muovono** tra un piano e l'altro, oppure si **fermano** ad un piano
  - Per passare dallo stato di movimento a quello di fermata si deve eseguire la regola **stop**
  - Per passare dallo stato di fermata a quello di movimento si deve eseguire la regola **partenza**
  - Dallo stato di movimento si passa ancora allo stato di movimento se in prossimità di un piano senza prenotazioni/chiamate
  - Dallo stato di fermata è possibile anche **cambiare direzione**
  - Assumiamo che il sistema sia inizializzato con gli ascensori **fermi** al **piano 0** e diretti verso l'**alto**

## Esempio: FSM del GM (1)



## Esempio: GM (2)

- Esecuzioni non vuote del FSM di ground model hanno la forma

$(\text{DEPART CONTINUE}^* \text{STOP})^+$   
 $(\text{CHANGE}(\text{DEPART CONTINUE}^* \text{STOP})^*)^*$

## Raffinamenti

### Raffinamento per passi successivi

- E' il secondo blocco concettuale usato nell'applicazione del metodo
- Consiste nell'ottenere a partire da una macchina più astratta una sua forma più raffinata (concreta)
- E' applicato iterativamente sino al punto di raffinamento considerato sufficiente dallo sviluppatore

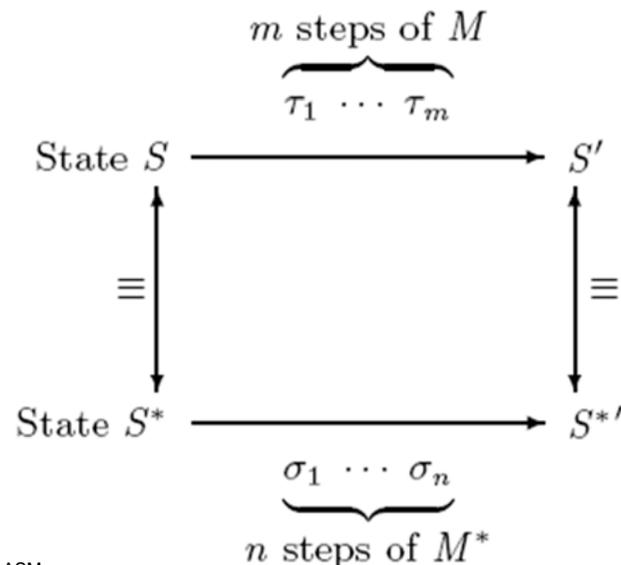
## Principio di sostituzione

- La sostituzione di un programma con un altro è accettabile se è **impossibile** per un utente rendersi conto della sostituzione
- Altri metodi applicano lo stesso principio
  - Spesso con varie restrizioni (forme del programma, operazioni sugli stati, interpretazioni sulle relazioni input-output, ...)
  - Queste restrizioni non valgono nel caso del metodo ASM

## Schema per il Raffinamento (1)

- Per raffinare una ASM  $M$  in una ASM  $M^*$  si devono definire:
  - le nozioni di **stato raffinato**; **stati di interesse**; **segmenti di computazione astratta**  $\tau_1, \tau_2, \dots, \tau_m$ , dove i  $\tau_i$ , rappresentano i singoli step di  $M$  e i corrispondenti **segmenti di computazione raffinata**  $\sigma_1, \sigma_2, \dots, \sigma_n$  in  $M^*$
  - le **corrispondenze** tra gli stati  $S$  di  $M$  e gli stati  $S^*$  di  $M^*$ ,
  - le location di interesse e la relazione di **equivalenza**  $\equiv$  dei dati nelle location di interesse

## Schema per il Raffinamento (2)



## Esempio di Raffinamento (1)

- Dettagliare le transizioni astratte
  - precisando le specifiche dei piani visitati
- E' un raffinamento 1-1:
  - ogni operazione astratta sugli ascensori è sostituita da un'operazione più dettagliata

## Esempio di Raffinamento (2)

- Dettagliamo le transizioni, specificando per ognuna il parametro  $L$ , che indica l'ascensore a cui la transizione è applicata
- Assumiamo che i piani siano ordinati
- La direzione  $dir(L)$  dell'ascensore  $L$  può essere **up** oppure **down**

## Esempio di Raffinamento (3)

- Sia  $floor(L)$  la funzione che indica il piano in cui si trova correntemente l'ascensore  $L$
- Definiamo  $MoveLift(L)$  come l'operazione che ha l'effetto immediato di aggiornare il  $floor(L)$ :

$MoveLift(L) =$

if  $dir(L) = up$  then  $floor(L) := floor(L) + 1$

if  $dir(L) = down$  then  $floor(L) := floor(L) - 1$

## Esempio di Raffinamento (4)

- Partenza:
  - dai requisiti deriva che un ascensore  $L$  abbandona il proprio stato di halt quando è **attirato** da un altro piano
  - Ogni  $L$  parte solo in direzione definita da  $dir(L)$ ; un  $L$  fermo parte sse è attirato secondo **Attracted** ( $dir(L), L$ )
  - quindi:

$Depart(L) = \text{if } Attracted(dir(L), L) \text{ then } (MoveLift(L))$

## Esempio di Raffinamento (5)

- Continue:
    - è analoga, inoltre se  $L$  **CanContinue**, aggiorna  $floor(L)$
    - quindi:
- $Continue(L) = \text{if } CanContinue(L) \text{ then } MoveLift(L)$

## Esempio di Raffinamento (6)

- Stop:

Stop(L)=

```
if not CanContinue(L)
then CancelRequest(dir(L), L)
```

## Esempio di Raffinamento (7)

- Change:

– assumiamo che un ascensore cambi direzione solo se attirato nella direzione **opposta** e se non è attirato secondo dir(L)

```
Change(L)= let d=dir (L) and d'= opposite (dir(L)) in
if not Attracted (d, L) and Attracted (d', L)
then {dir(L):= d', CancelRequest(d',L)}
```

## Esempio di Raffinamento (8)

- Si ottiene la seguente ASM:

Lift(this) =

```
FSM(halting, Depart(this), moving)
```

```
FSM(moving, Continue(this), moving)
```

```
FSM(moving, Stop(this), halting)
```

```
FSM(halting, Change(this), halting)
```

- Nota: this suggerisce un'implementazione o.o. della classe Lift, con metodi Depart, Continue,

Stop e Change

## Prova

- Esercizio: Si mostri che l'ASM definita precedentemente soddisfa i requisiti dati

## Ulteriore Raffinamento (1)

- E' possibile effettuare un ulteriore raffinamento 1:1
  - raffinando i dati (le sentinelle delle regole)
  - raffinando le operazioni (la macro CancelRequest)

## Ulteriore Raffinamento (2)

- Osserviamo che le sentinelle possono essere derivate da
  - una funzione interna hasToDeliverAt (L, floor)
    - si suppone che inizialmente restituisca false
  - una funzione esterna existsCallFromTo(floor, dir)
    - si suppone che inizialmente restituisca false
    - è false per le coppie di parametri <ground, down>, <top, up>

## Ulteriore Raffinamento (3)

- L deve visitare un piano se si preme il tasto di quel piano nella pulsantiera di L
  - Quindi
- HasToVisit(L, floor)  $\Leftrightarrow$  hasToDeliverAt (L, floor)
- or
- esiste dir t.c. existsCallFromTo(floor, dir)

## Ulteriore Raffinamento (4)

Attracted (d, L)  $\Leftrightarrow$

d=up and esiste f>floor t.c. hasToVisit (L, f), or  
d = down and esiste f<floor t.c. hasToVisit (L, f)

CanContinue (L)  $\Leftrightarrow$

Attracted (dir(L), L) and  
not hasToDeliverAt(L, floor(L)) and  
not existsCallFromTo(floor(L), dir(L))

## Ulteriore Raffinamento (5)

Quindi

CancelRequest (d, L) =

hasToDeliverAt(L, floor(L)):=false

existsCallFromTo(floor(L), dir):=false

## Prova

La precedente ASM (LIFT) soddisfa i requisiti

## Principi per la prova di correttezza del raffinamento

- Obiettivo: mostrare che una implementazione  $S^*$  soddisfa una proprietà desiderata  $P^*$
- Passi:
  - costruire un modello astratto  $S$
  - dimostrare che una forma astratta  $P$  della proprietà  $P^*$  vale secondo un insieme  $A$  di appropriate assunzioni su  $S$
  - mostrare che  $S$  è raffinato correttamente da  $S^*$  e che le  $A$  valgono anche in  $S^*$