Safety-Critical Systems 4: Engineering of Embedded Software Systems

© Jan Bredereke
University of Bremen

WS 2002/03

0. Introduction

Topic of This Lecture

intersection of:

- engineering
- embedded systems
- software systems

Engineering

- the disciplined use of science, mathematics and technology to build useful artefacts
- engineers design by means of documentation
  - key step: design validation
  - maintenance requires good documentation

Embedded Systems

- definition: an embedded computer system is considered a module in some larger system
- some distinguishing characteristics:
  - designer not free to define interface
  - interface constraints may be strict and arbitrary, but we can't ignore them
  - interfaces will change during development

Examples of Embedded Systems

- computer in autonomous wheelchair
  constraints: devices
  sensor data
  physics of wheelchair
- telephone switching system
  constraints: other company's switches
  own older switches
  international standards
  telephone number rules

The Safety-Critical Systems Lecture Series

SCS1: Basic concepts - problems - methods - techniques (SoSe02)

SCS2: Management aspects - standards - V-Models - TQM - assessment - process improvement (SoSe01, SoSe03)

SCS3: Formal methods and tools - model checking - testing - partial verification - inspection techniques - case studies (WiSe01/02)

SCS4: Engineering of Embedded Software Systems

Overview of SCS4

1. *rigorous description* of requirements
   1.1 system requirements
   1.2 software requirements
2. *what information* should be provided in computer system documentation?
3. *decomposition* into modules
   3.1 the criteria to be used in decomposing systems into modules
   3.2 time and space decomposition of complex structures
   3.3 designing software for ease of extension and contraction
4. design of the *module interfaces*

5. *families* of systems
   5.1 motivation: maintenance problems in telephone switching
   5.2 families of programs
   5.3 families of requirements

Style of This Course

- lecture 2 SWS (Vorlesung)
  - "This is obvious, isn't it?"
- seminar 2 SWS (Übung)
  - "Oops, applying it here is difficult!"

Web Page of Lecture

`www.tzi.de/agbs/lehre/ws0203/scs4`

available for download:

- slides
- assignments
- announcements
- links
- ...

Text for Reading

- lecture based on a number of research papers
- references will be given during course
  - mostly, not online :-(
  - important ones available for copying from secretary

Mark ("Schein")

- n assignments during term, $7 \leq n \leq 14$
- assignments can be solved in *groups of two*
- $n-1$ assignments must be *handed in*
- *average of* $n-1$ best marks must be $\geq 60\%$
- *oral exam ("Fachgespräch")* at end of term
  - 15-20 min
  - in the groups of two
  - individual marks

1. Rigorous Description of Requirements

Text for Chapter 1

[PaMa95] Parnas, D. L. and Madey, J. *Functional documents for computer systems*. Sci. Comput. Programming **25**(1), 41–61 (Oct. 1995).

Four-variable model, structure of requirements documentation and software documentation.

[Pet00] Peters, D. K. *Deriving Real-Time Monitors from System Requirements Documentation*. PhD thesis, McMaster Univ., Hamilton, Canada (Jan. 2000).

Most current version of four-variable model and tabular notation. (Is also on testing).
Relevant: Chapters 1.1, 5, Appendix A

Additional Background for Chapter 1

[vSPM93] van Schouwen, A. J., Parnas, D. L., and Madey, J. *Documentation of requirements for computer systems*. In "IEEE Int'l. Symposium on Requirements Engineering – RE'93", pp. 198–207, San Diego, Calif., USA (4–6 Jan. 1993). IEEE Comp. Soc. Press.

Example for the four-variable approach (water level monitoring system).

[LaRö01] Lankenau, A. and Röfer, T. *The Bremen Autonomous Wheelchair – a versatile and safe mobility assistant*. IEEE Robotics and Automation Magazine, "Reinventing the Wheelchair" **7**(1), 29–37 (Mar. 2001).

General description of the Bremen autonomous wheelchair "Rolland".

The Role of Documentation in Computer System Design

- professional engineer:
  - makes plan on paper
  - analyses plan thoroughly
  - then builds system, using plan
- engineer revising the system:
  - understands system through old plan
  - changes plan
  - analyses plan thoroughly
  - then builds system, using plan

- Computer hardware is made this way.
- Computer software usually is not.
- But standard engineering practice can be applied, too.
- Documentation
  - as a design medium
  - input to analysis
  - input to testing
  - facilitates revision or replacement

Education of Engineers Can't Start Too Early. . .

from a text book on engineering:

1.1 System Requirements

How Can We Document System Requirements?

- identify the relevant environmental quantities
  - physical properties
    - ▷ temperatures
    - ▷ pressures
  - positions of switches
  - readings of user-visible displays
  - wishes of a human user
- represent them by mathematical variables

- define carefully the association of env. quantities and math. variables
- specify a relation on the math. variables

Functions of Time

- env. quantities q_i described by functions of time
- types of values of env. quantities: $q_i \in Q_i$
- environmental state function:
  $S : \mathbb{R} \to Q_1 \times Q_2 \times \ldots \times Q_n$
- set of possible env. states:
  $St =_{\text{df}} Q_1 \times Q_2 \times \ldots \times Q_n$

Example: Electronic Thermometer

→ blackboard. . .

Monitored vs. Controlled Quantities

- controlled quantities:
  their value may be required to be changed by the system
- monitored quantities:
  shall affect the system behaviour
- some quantities are both
- time: is a monitored quantity
  (in real-time systems)

- monitored state function:
  $\underset{\sim}{m}^t : \mathbb{R} \to Q_1 \times Q_2 \times \ldots \times Q_i, \qquad 1 \le i \le n$
- controlled state function:
  $\underset{\sim}{c}^t : \mathbb{R} \to Q_j \times Q_{j+1} \times \ldots \times Q_n, \qquad 1 \le j \le n$
- $j \le i + 1$
- environmental state function: $(\underset{\sim}{m}^t, \underset{\sim}{c}^t)$
- set of all $\underset{\sim}{m}^t$: M
- set of all $\underset{\sim}{c}^t$: C
- "behaviour": an S for a single execution

The Relation NAT

- constraints on the environmental quantities
- constraints by nature, by previously installed systems
- is part of the requirements document
- validity is responsibility of customer

- $\mathrm{NAT} \subseteq \mathrm{M} \times \mathrm{C}$
- $\mathrm{domain}(\mathrm{NAT}) = \{\underset{\sim}{m}^t \mid \underset{\sim}{m}^t$ allowed by env. constraints$\}$
  - if $\underset{\sim}{m}^t \notin \mathrm{domain}(\mathrm{NAT})$ then designer may assume that these values never occur
- $\mathrm{range}(\mathrm{NAT}) = \{\underset{\sim}{c}^t \mid \underset{\sim}{c}^t$ allowed by env. constraints$\}$
  - if $\underset{\sim}{c}^t \notin \mathrm{range}(\mathrm{NAT})$ then system cannot make these values happen
- $(\underset{\sim}{m}^t, \underset{\sim}{c}^t) \in \mathrm{NAT}$ iff environmental constraints allow that $\underset{\sim}{c}^t$ are controlled quantities if $\underset{\sim}{m}^t$ are monitored quantities
- NAT usually not a function
  - the system should have some choice

The Relation REQ

- further constraints on the environmental quantities
- constraints by system
- is part of the requirements document
- validity is responsibility of system designer

- $\mathrm{REQ} \subseteq \mathrm{M} \times \mathrm{C}$
- $\mathrm{domain}(\mathrm{REQ}) \supseteq \mathrm{domain}(\mathrm{NAT})$
  $= \{\underset{\sim}{m}^t \mid \underset{\sim}{m}^t$ allowed by env. constraints$\}$
- $\mathrm{range}(\mathrm{REQ}) = \{\underset{\sim}{c}^t \mid \underset{\sim}{c}^t$ allowed by correct system$\}$
- $(\underset{\sim}{m}^t, \underset{\sim}{c}^t) \in \mathrm{REQ}$ iff system should permit that $\underset{\sim}{c}^t$ are controlled quantities if $\underset{\sim}{m}^t$ are monitored quantities
- REQ usually not a function
  - one can tolerate "small" errors in the values of controlled quantities

Contract

- REQ states what the system designer must provide
- NAT states what the customer must provide

Black-Box View

- the requirements document is entirely in terms of environmental quantities
- no reference to internal quantities
- no reference to internal state, only to the history of env. quantities
- $\Rightarrow$ no restriction on system designer

Specifying Behaviour

what's next?

- modes and mode classes
- conditions, events
- four-variable approach for system design and software requirements
- tabular notation

Modes and Mode Classes, Informally

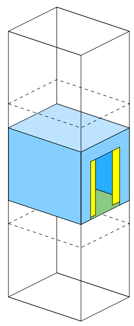
Definition 1 (Mode, informally)
An (environmental) mode is a set of (environmental) states.

Definition 2 (Mode Class, informally)
A mode class is a partitioning of the state space.

Discussion of Modes etc.

- there may be several mode classes
- system is always in one mode of every mode class
- mode class and its modes may be defined by a transition table
- one state change may imply two mode changes
  - no "simultaneous events"
- if time is monitored, the system never returns into the same state
  - modes are handy for equivalence classes of states

Example: Lift Controller



Lift Controller: Relevant Environmental Quantities

- height of lift
- elevation motor command
- position of doors
- door motor command
- (buttons left out for simplicity here)

Lift Controller: Environment Variables

Variable	mon.	ctrl.	Description	Value Set	Unit	Notes
^{m}height	•		height of lift	$\mathbb{R}$	m	1
^{c}elevMotorCommand		•	elevation motor command	$\{^C\text{up}, {}^C\text{off}, {}^C\text{down}\}$	—	
^{m}doorPos	•		position of doors	$\mathbb{R}$	m	2
^{c}elevMotorCommand		•	door motor command	$\{^C\text{open}, {}^C\text{off}, {}^C\text{close}\}$	—	

Notes
1. The height is relative to the lowest position physically possible, upward is positive.
2. This is how far the doors are opened. 0 m means entirely closed, positive means open.

Lift Controller: the Relation NAT

what to state rigorously (not done here):

- height is ≥ 0 m and $\leq$ max. height
- the acceleration and deceleration of the lift is bounded ($\rightarrow$ use differential equations)
- door position is ≥ 0 m and $\leq$ max. width
- ...

Conditions

Definition 3 (Condition)
A condition is a function $\mathbb{R} \rightarrow \mathbb{B}$,
defined in terms of the env. state function.
It is finitely variable on all intervals of system operation.
Definition 4 (Cnd)
Cnd is the tuple of all conditions.
We assume an order on the conditions.
We assume Cnd to be finite.

Lift Controller: Conditions

Name	Condition
^{p}doorClosed	^{m}doorPos $= 0$ m
^{p}at1stFloor	$\vert{}^m$height $- 0.5$ m$\vert \leq 1$ cm
^{p}at2ndFloor	$\vert{}^m$height $- 4.5$ m$\vert \leq 1$ cm
^{p}at3rdFloor	$\vert{}^m$height $- 8.5$ m$\vert \leq 1$ cm

Cnd $= ({}^p$doorClosed$, {}^p$at1stFloor$, {}^p$at2ndFloor$, {}^p$at3rdFloor$)$

Events

Definition 5 (Event)
An event e, *is a pair,* (t, c), *where*
$e.t \in \mathbb{R}$ *is a time at which one or more conditions change value and*
$e.c \in \{T, F, @T, @F\}^n$ *indicates the status of all conditions at* $e.t$, *as follows:*

$e.c[i]$	p_i
T	${}^{\backprime}p_i(e.t) \wedge p_i{}'(e.t)$
F	$\neg{}^{\backprime}p_i(e.t) \wedge \neg p_i{}'(e.t)$
@T	$\neg{}^{\backprime}p_i(e.t) \wedge p_i{}'(e.t)$
@F	${}^{\backprime}p_i(e.t) \wedge \neg p_i{}'(e.t)$

Event Space

Definition 6 (Event Space)
The event space is the set of all possible events:
$EvSp =_{df} \mathbb{R} \times \{T, F, @T, @F\}^n$

- any particular finite duration behaviour defines a finite set of events Ev $\subset$ EvSp

Lift Controller: Events

- (5 s, (F, @T, F, F))
- (7 s, (@T, T, F, F))
- (20 s, (@F, T, F, F))
- (22 s, (F, @F, F, F))
- (29 s, (F, F, @T, F))
- ...

History

- we are often interested in the values of the conditions for a specific interval of time
- a history is
  - the set of initial values for the conditions and
  - the sequence of events in the time interval
- (formal definition omitted here)

Modes and Mode Classes

Definition 7 (Mode Class)
A (environmental) mode class is an equivalence relation on possible histories, such that:
if $\mathrm{MC}(H_1, H_2)$ *and*
if $\hat{H}_1$ *and* $\hat{H}_2$ *are the extensions of* H_1 *and* H_2
by the same event,
then $\mathrm{MC}(\hat{H}_1, \hat{H}_2)$.
Definition 8 (Mode)
An (environmental) mode is one such equivalence class.

Lift Controller: Mode Classes

- some useful mode classes:
  - ^{Cl}door: ^{Md}doorClosed, ^{Md}doorOpen
  - ^{Cl}floor: ^{Md}in1stFloor, ^{Md}in2ndFloor, ^{Md}in3rdFloor
  - ^{Cl}atFloor: ^{Md}atAFloor, ^{Md}betweenFloors
- definition of mode classes:
  - through conditions
  - by transition tables

  (see later)

Tabular Notation

- tabular notations often useful to represent functions in computer system documentation
- extensive work on different table formats exists
- precise semantics has been defined for these table formats
- one format specifically for mode transition tables

Lift Controller: the Relation REQ

- conditions defined in terms of (monitored) variables
- event classes defined in terms of conditions
- mode classes defined in terms of event classes
- controlled variables defined in terms of mode classes

^{Cl}floor:

Mode	Event Class	New mode
^{Md}in1stFloor	@T(^{p}at2ndFloor)	^{Md}in2ndFloor
^{Md}in2ndFloor	@T(^{p}at1stFloor)	^{Md}in1stFloor
	@T(^{p}at3rdFloor)	^{Md}in3rdFloor
^{Md}in3rdFloor	@T(^{p}at2ndFloor)	^{Md}in2ndFloor

- the mode remains the same when between floors

"Simultaneous" Events

- modes of a mode class must be disjoint
- $\rightarrow$ event classes for one mode must be disjoint
- event expressions can comprise more than one event
- assume that causally independent changes of conditions never occur at exactly the same time ($t \in \mathbb{R}$)
- watch out for condition changes that are causally related!

Piecewise Continuous Behaviour

- often, environmental quantities have piecewise continuous behaviour over time
  - height of lift
  - position of lift door
- each continuous piece can be described well by a differential equation
- switching from piece to piece can be described well by mode changes

Lift Controller: Piecewise Continuous Behaviour

one of the constraints by NAT:

$\frac{d}{dt}{}^{m}$height $=$

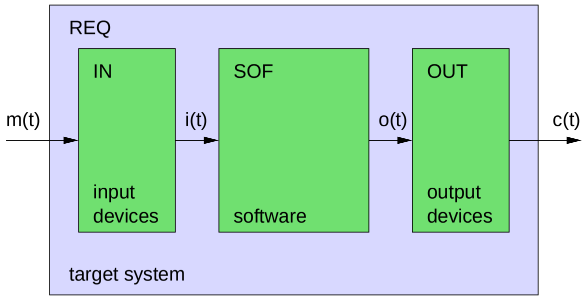
inmode(^{Md}up)	^{C}liftSpeed
inmode(^{Md}standStill)	0 cm/s
inmode(^{Md}down)	$-^{C}$liftSpeed

1.2 Software Requirements

System Design

- decisions on what to do in hardware/software
- results in:
  - ◦ hardware requirements
  - ◦ software requirements

The Four-Variable Approach for System Design and Software Requirements



Input and Output Quantities

- input state function:
  $\underset{\sim}{i}^t : \mathbb{R} \to \mathrm{I}_1 \times \mathrm{I}_2 \times \ldots \times \mathrm{I}_n$
- output state function:
  $\underset{\sim}{o}^t : \mathbb{R} \to \mathrm{O}_1 \times \mathrm{O}_2 \times \ldots \times \mathrm{O}_m$
- set of all $\underset{\sim}{i}^t$: I
- set of all $\underset{\sim}{o}^t$: O
- behaviour required of
  - ◦ the input devices: $\mathrm{IN} \subseteq \mathrm{M} \times \mathrm{I}$
  - ◦ the output devices: $\mathrm{OUT} \subseteq \mathrm{O} \times \mathrm{C}$
  - ◦ the software: $\mathrm{SOF} \subseteq \mathrm{I} \times \mathrm{O}$

Software Acceptability

- the software requirements SOFREQ are determined completely by REQ, NAT, IN, and OUT
  - ◦ $((\mathrm{IN} \cdot \mathrm{SOFREQ} \cdot \mathrm{OUT}) \cap \mathrm{NAT}) = \mathrm{REQ}$
  - ◦ SOFREQ usually difficult to calculate precisely
- a software SOF is acceptable if
  SOF with IN and OUT and NAT imply REQ:
  $((\mathrm{IN} \cdot \mathrm{SOF} \cdot \mathrm{OUT}) \cap \mathrm{NAT}) \subseteq \mathrm{REQ}$
  - ◦ some design decisions make life easier
    - ▷ remove some non-determinism
    - ▷ $\mathrm{SOF} \subseteq \mathrm{SOFREQ}$
  - ◦ SOF must still be acceptable

First Application of Four-Variable Method

- software cost reduction project (SCR)
  - ◦ developed the method
  - ◦ US Naval Research Laboratory (NRL)
- specification of the complete software requirements for the A-7 aircraft's TC-2 on-board computer
  - ◦ reverse-engineering of existing system
  - ◦ with help from domain experts (pilots, . . .)
- maintained over lifetime of system
  - ◦ first release: Nov. 1978
  - ◦ end of project: Dec. 1988
- 473 pages

Example: Autonomous Wheelchair "Rolland"

- Univ. of Bremen, AG B. Krieg-Brückner (Thomas Röfer, Axel Lankenau, . . .)
- joystick-to-motor line wiretapped
- ring of sonar sensors
- safety module
- driving assistant
  - ◦ turning on the spot skill
  - ◦ obstacle avoidance skill
  - ◦ . . .

Rolland: Specification of Safety-Relevant Behaviour

- very recent research work on "mode confusion" problems
- requirements documented by A. Lankenau, J. Bredereke
- reverse-engineering work
- language: CSP
  - ◦ different formalism
  - ◦ model-checking tool available
  - ◦ CSP starts out with events, not variables
  - ◦ otherwise same software engineering approach used
- slides: presentation ignores CSP

Rolland: Relevant Environmental Quantities

- the joystick command
- the wheelchair motors command
- the actual wheelchair motors status
- location of the obstacles near the wheelchair

Rolland: Environment Variables

Variable	mon.	ctrl.	Description	Type	Notes
^{m}t	•		current time	$\mathbb{R}$	
^{m}joystickCommand	•		the user intended motion as indicated with the joystick	^{t}JoystickCommandVector	
^{c}motorsCommand		•	command for the wheelchair motors	^{t}MotorsCommandVector	
^{m}motorsActual	•		the actual motors status of the wheelchair	^{t}MotorsCommandVector	
^{m}obsLoc	•		location of relevant obstacles	^{t}obstacleLocs	
^{m}orientation	•		the current orientation of the wheelchair	^{t}orientationRange	1

Notes

1. The orientation is relative to the world (inertial system). At program start, the orientation is 0°.

Rolland: Environment Variable Types

Type	Description	Values	Unit
^{t}JoystickCommandVector	a joystick command vector (i,d). i: fraction of max. joystick inclination, d: direction of the joystick inclination	^{t}inclinationRange × ^{t}orientationRange	(%, °)
^{t}inclinationRange	fraction of max. joystick inclination	$\{x \in \mathbb{R} \mid 0 \leq x \leq 100\}$	%
^{t}orientationRange	a direction. 0: straight ahead 90: left 180: straight back -90: right	$\{x \in \mathbb{R} \mid -180 < x \leq 180\}$	°

Type	Description	Values	Unit
^{t}MotorsCommandVector	A command vector (s,a) sent to the motors as target value. s: speed value, restricted by physical limitations of the wheelchair, a: angle of the wheelchair's steering wheels	^{t}speedRange × ^{t}steeringAngleRange	(cm/s, °)
^{t}speedRange	physical wheelchair speed range (167 cm/s is 6 km/h)	$\{x \in \mathbb{R} \mid -167 \leq x \leq 167\}$	cm/s
^{t}steeringAngleRange	angle of steering wheels of wheelchair. -60: right 0: straight 60: left	$\{x \in \mathbb{R} \mid -60 \leq x \leq 60\}$	°
^{t}obstacleLocs	...	...	...

(the rather complex type ^{t}obstacleLocs is omitted in the slides)

Rolland: Observations

- precise link between environmental quantities and mathematical variables
  - definitions in rigorous prose
  - explicit units
  - explicit meaning of individual values of a range
- tabular format suitable
- duplication of description avoided

Rolland: Conditions and Events

- simple for Rolland
- not specified separately
- specified in-place in the relations (see later)

Rolland: the Relation NAT

complete description would comprise:

- the wheelchair obeys to commands after a delay
  - acceleration/deceleration
  - steering
- obstacles don't move by themselves
- obstacles are always visible for the sonar sensors

- simplified specification for case study:

$$\exists t_d \in (0 \ldots {}^C\text{maxDelMot}] \;.\; {}^m\text{motorsActual} = {}^c\text{motorsCommand}({}^m\text{t} - t_d)$$

- restrictions of value ranges already specified by types
- convention: if omitted, ^{m}t is parameter implicitly

Rolland: the Relation REQ

- was specified in case study only implicitly
  - because of reverse-engineering approach
- explicitly: IN, SOF, OUT, and NAT
- we can assume $\mathrm{SOFREQ} = \mathrm{SOF}$ and then derive $\mathrm{REQ} = ((\mathrm{IN} \cdot \mathrm{SOFREQ} \cdot \mathrm{OUT}) \cap \mathrm{NAT})$

Rolland: Input Variables

Input Variable	Description	Type	Notes
^{i}joystickUnitCommand	the user intended motion as indicated with the joystick	^{t}JoystickUnitCommandVector	
^{i}motorsUnitActual	the actual motors status of the wheelchair	^{t}MotorsUnitCommandVector	
^{i}obsLoc	location of relevant obstacles	^{t}obstacleLocsMap	1
^{i}orientation	the current orientation of the wheelchair	^{t}odoOrientationRange	2

Notes

1. This does not include obstacles that cannot be detected by the wheelchair's sonar sensors, because of their known technical limitations (surface structure dependance, objects visible only at sensor level, etc.)
2. The orientation is relative to the world (inertial system). At program start, the orientation is $0°$. This information is only reliable over short distances due to odometry drift.

Rolland: Input and Output Variable Types

Type	Description	Values	Unit
^{t}MotorsUnitCommandVector	A command vector (s,r) interpreted by the motors unit. s: speed value, restricted by safety and comfort limitations of the wheelchair, r: curve radius of the wheelchair's steering wheels	^{t}SpeedCommandRange $\times$ ^{t}RadiusRange	(cm/s, cm)
^{t}JoystickUnitCommandVector	a command vector (s,r) containing the interpreted joystick command, interpretation as above	^{t}MotorsUnitCommand-Vector	(cm/s, cm)
^{t}SpeedCommandRange	speed range used for target commands (coming from the joystick and sent to the motor) (84 cm/s is 3 km/h)	$\{x \in \mathbb{N} \mid -42 \leq x \leq 84\}$	cm/s

Type	Description	Values	Unit
^{t}RadiusRange	curve radius range < 0: right curve > 0: left curve 0: straight other values between -50 and $+50$ are physically impossible and are interpreted as -50 and $+50$, respectively	$\mathbb{N}$	cm
^{t}odoOrientationRange	a direction, as computed by odometry.	$\{x \in \mathbb{N} \mid -180 < x \leq 180\}$	°
^{t}obstacleLocsMap	...	...	...

(the rather complex type ^{t}obstacleLocsMap is omitted in the slides)

Rolland: Output Variables

Output Variable	Description	Type	Notes
^{o}motorsUnitCommand	the command for the wheelchair motor unit	^{t}MotorsUnitCommandVector	

Rolland: the Relation IN

^{i}joystickUnitCommand =

^{m}joystickCommand.d $> 90 \vee$ ^{m}joystickCommand.d < -90	(round(^{m}joystickCommand.i$/100 \cdot -42$), calcRadius(calcSteeringAngle(^{m}joystickCommand.d)))
$\neg$(^{m}joystickCommand.d $> 90 \vee$ ^{m}joystickCommand.d < -90)	(round(^{m}joystickCommand.i$/100 \cdot 84$), calcRadius(calcSteeringAngle(^{m}joystickCommand.d)))

Note: round, calcSteeringAngle, and calcRadius are functions defined in the Dictionary and omitted in the slides.

^{i}motorsUnitActual =

^{m}motorsActual.s $\geq -42 \wedge$ ^{m}motorsActual.s ≤ 84	(round(^{m}motorsActual.s), calcRadius(^{m}motorsActual.a))
^{m}motorsActual.s < -42	(-42, calcRadius(^{m}motorsActual.a))
^{m}motorsActual.s > 84	(84, calcRadius(^{m}motorsActual.a))

^{i}orientation = round(^{m}orientation)

^{i}obsLoc = ...

Rolland: the Relation OUT

^{c}motorsCommand = (^{o}motorsUnitCommand.s, calcMotorSteeringAngle(^{o}motorsUnitCommand.r))

Rolland: the Relation SOF

- complex behaviour, see specification in CSP
- specify output variables in terms of input variables
- use mode classes as appropriate