Towards a Formally Grounded Development Method

Christine Choppy [1] and Gianna Reggio [2]

[1] LIPN, Institut Galilée - Université Paris XIII,

[2] DISI Università di Genova - Italy

Outline and motivation

- Write relevant, legible, useful specifications of the systems to be developed
- Informal notations (graphics)/formal (semantics)
- Companion user method helping to understand the system to be developed (different from helping to use the proposed formalism)
- Accomodate different natures of systems
- The best of both worlds !?

	FORMAL	INFORMAL
notation	not very friendly (exotic)	very friendly, visual
notation	rigid, overhead	flexible, adaptable
learning	time, background	short(?)
case studies	simple (?)	real common app

Outline and motivation (2)

Methods taking into account:

- a software item:
  - *a simple dynamic system*
  - *a structured dynamic system*
  - *a data structure*
- two specification techniques: *property-oriented, model-oriented* (constructive)
- CASL and CASL-LTL specifications

Illustration on case studies

To be used

- for requirement specifications
- in combination with structuring concepts as (Jackson's) problem frames

Complementary related works

- How to write readable CASL specifications, avoiding semantic pitfalls
  http://www.brics.dk/Projects/CoFI
  - Roggenbach and Mossakowski for the basic data types library
  - Bidoit and Mosses in the CASL user's manual
- Bidoit and Hennicker [e.g. FOSSACS02] explore the use of observability concepts which are found to be useful and relevant for writing specifications, and the combined use of constructors and observers
- Blanc [PhD 2002, Cachan] proposes guidelines for the iterative and incremental development of specifications
- Choppy and Reggio [WADT99] propose to help requirement analysis by generating CASL and CASL-LTL skeletons associated with Jackson's problem frames (used as structuring concepts to start the problem analysis)
- Choppy and Heisel [WADT02] propose to go on with using the structuring concepts provided by architectural styles to support design specifications and explore the combination with the problem frames used to begin with

CASL and CASL-LTL

- CASL (Common Algebraic Specification Language)
  partial ops, datatypes declarations, union, extension
  free construct, generic specifications

- CASL-LTL a simple system is considered as a labelled transition system (lts):
  labels, states and transition relation
  Labelled Transition Logic [Astesiano, Reggio, Costa, TCS97]

  dsort *st* **label** *lab* stands for **sorts** *st*, *lab*
  pred $\_\_ \xrightarrow{\_\_} \_\_ : st \times lab \times st$

  temporal logic (branching, CTL like) used to express properties of the dynamic systems in terms of their paths or sequences of transitions, e.g. :
  $in\_any\_case(S, \pi)$ or $in\_one\_case(S, \pi)$
  when a formula holds on the first state of a path,
  at the first label of a path, eventually, always . . . .

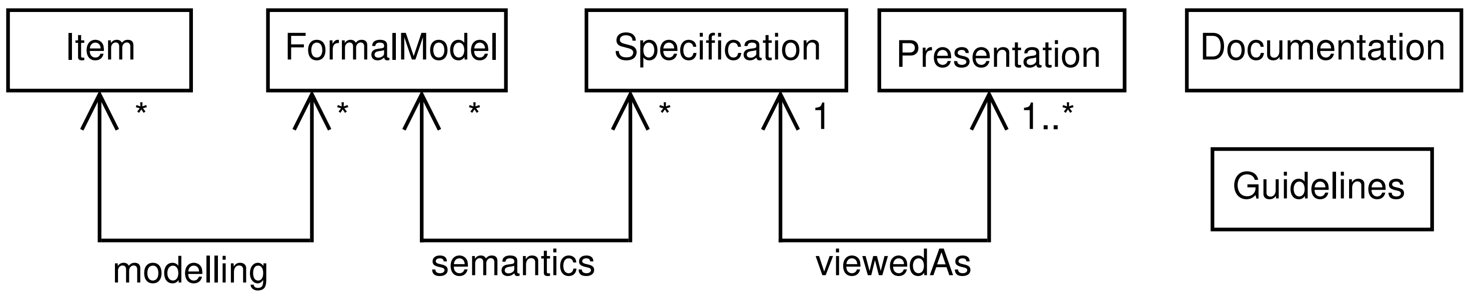
Case Study: a lift system

- a *lift plant* (the cabin, the motor moving it, the doors at the various floors)
- the *controller* (some software automatically controlling the lift functioning)
- the *users*

- *sensors* (e.g., cabin position, doors at floors, motor working status)
- *orders* (e.g., open/close the doors, move up/down/stop motor)
- users enter or leave the cabin . . .

Ingredients for a generic specification method

adapted from Astesiano, Reggio, TCS 2000.



1 - Items that will be specified

2 - Formal models of the items

3 - Modelling

4 - Specification

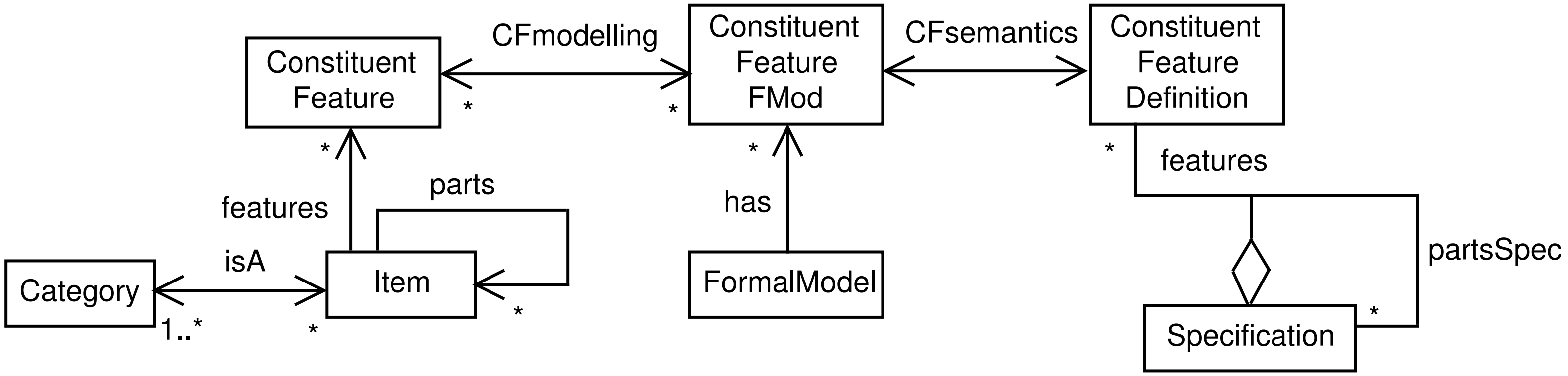
5 - Semantics

6 - Presentation

7 - Documentation

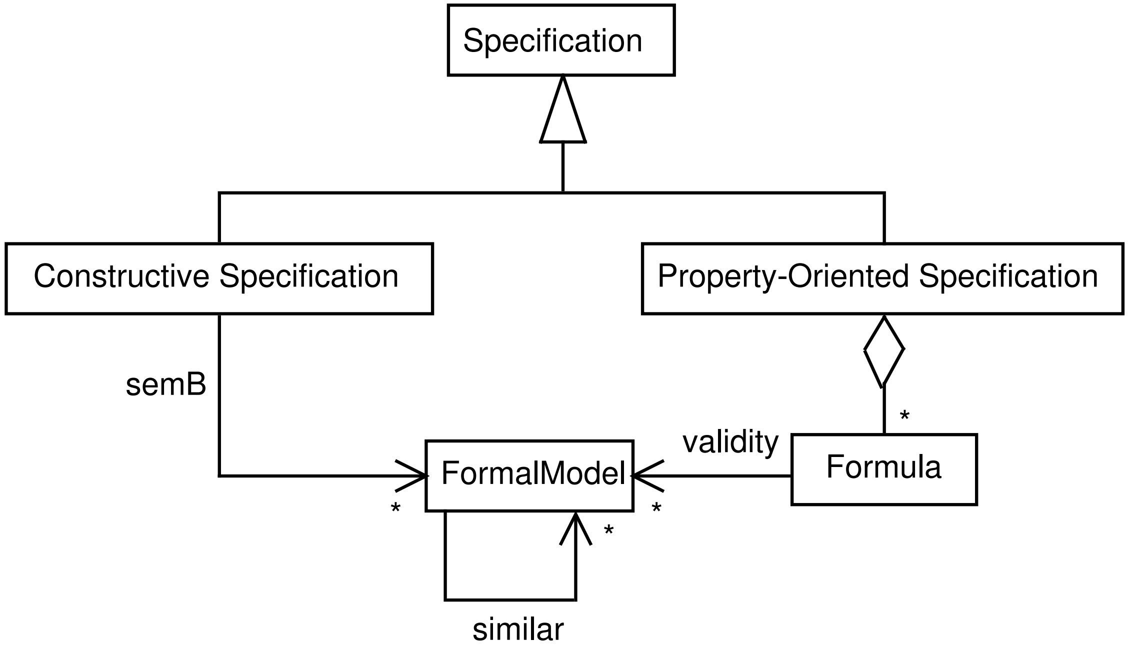
8 - Guidelines

Items



- *structured* (parts)
- characterized by constituent features of different *kinds*

Property vs Model oriented

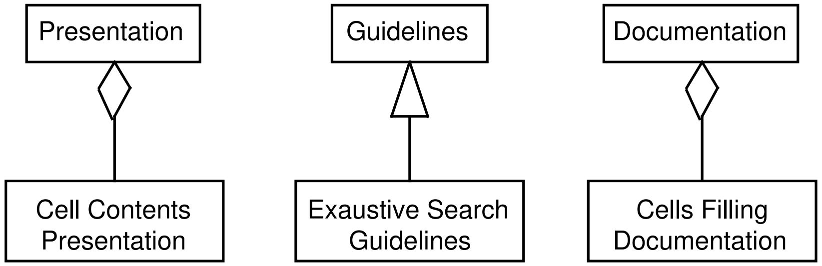


- Property-oriented (axiomatic) : “relevant” properties expressed
- Model-oriented (constructive) : exhibit a prototype . . .

for: *simple dynamic systems, structured dynamic systems, data structures*

“6” specification methods with common parts.

A General Property-oriented Specification Method (GPSm)



Find: parts, constituent features, express properties (cell filling, presentation).

		KIND$_1$				KIND$_k$		
		CF^1_1	.....	CF^1_{n1}	.....	CF^k_1	.....	CF^k_{nk}
KIND$_1$	CF^1_1							
	.....							
	CF^1_{n1}							
	.....							
KIND$_k$	CF^k_1							
	.....							
	CF^k_{nk}							

Outline

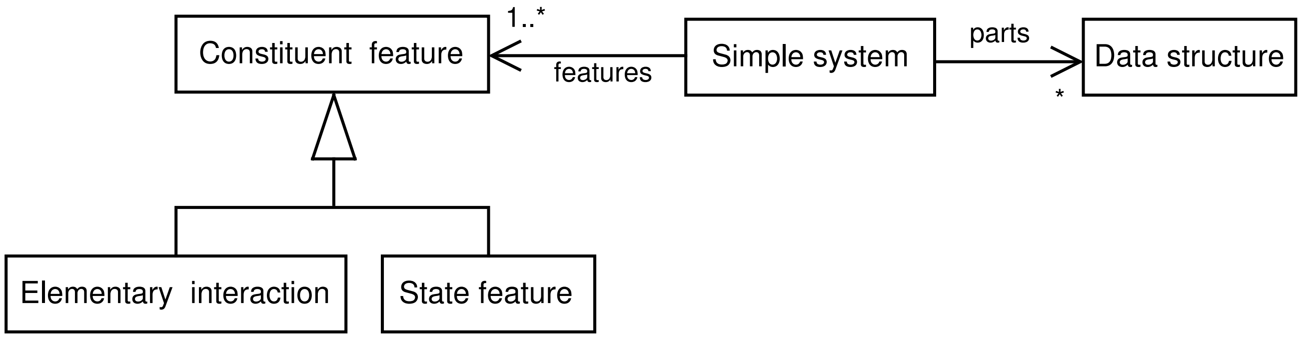
Methods taking into account:

- a software item:
  - a simple dynamic system
  - *a structured dynamic system*
  - *a data structure*
- two specification techniques: *property-oriented, model-oriented* (constructive)
- CASL and CASL-LTL specifications

Illustration on case studies

- in combination with structuring concepts as (Jackson's) problem frames

A Simple System



A dynamic system without any internal components cooperation.

A labelled transition system.

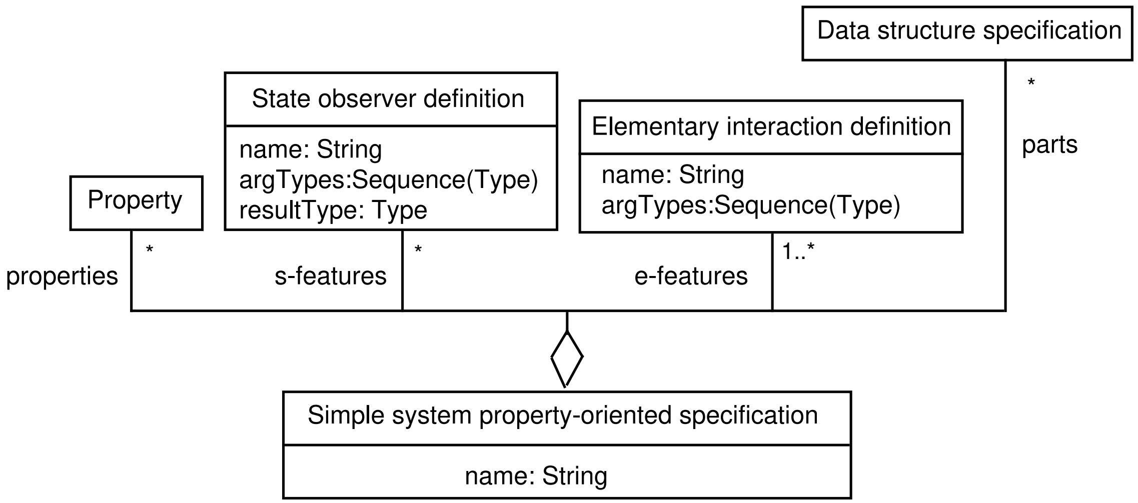
Constituent features:

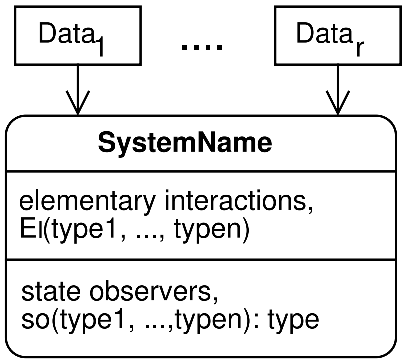
- state constituent features

- label: elementary interactions of different types

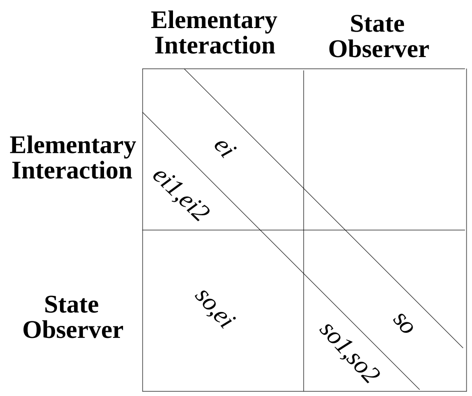
Parts: data structures

Property-oriented specifications (Simple systems)



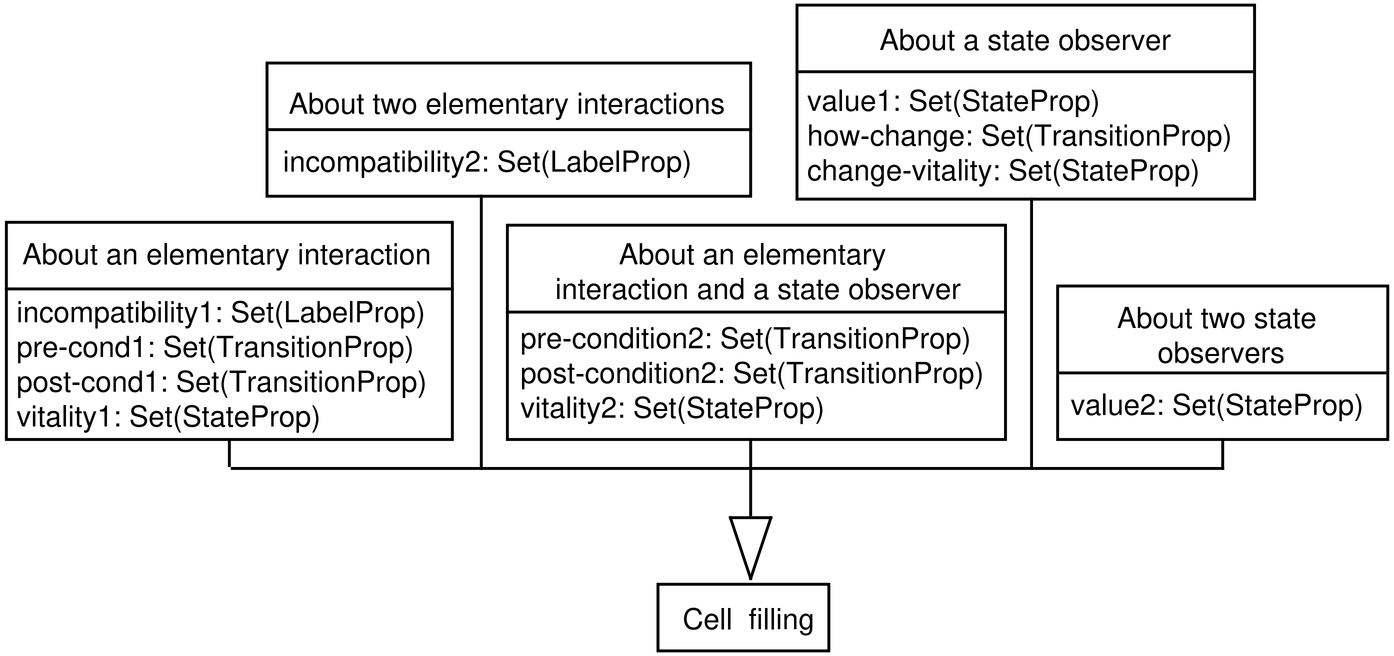


Visual presentation



Cell filling

Cell schemata (Simple system/Property-oriented)



Each cell may contain several properties of different nature. Properties on:

- labels (incompatibilities between elementary interactions under some condition)
- states (state observers properties where path properties may appear)
- transitions (conditions on source and target state observers).

Cell: About a state observer (*so*) -(Simple/Property)

value1 (state property) The results of the observation made by *so* on a state must satisfy some conditions.

cond, where *so* must appear in *cond*

how-change (transition property) If the observed value changes during a transition, then some condition on the source and target state (the old and the new value) holds, and some elementary interactions must belong to the transition label.

if *so*(*arg*) = v_1 **and** *so*'(*arg*) = v_2 **and** $v_1 \neq v_2$ **then** *cond*(v_1,v_2,*arg*) **and** $ei_1, \ldots, ei_n$ **happen**

change-vitality (state property) If a state satisfies some condition, then the observed value will change in the future.

if *cond*(v_1,v_2,*arg*) **and** *so*(*arg*) = v_1 **and** $v_1 \neq v_2$ **then in any case eventually** *so*(*arg*) = v_2

Note: “at least in a case” (instead of “in any case”) or “next” (instead of “eventually”) are possible.

Cell: About an elementary interaction (*ei*) -(Simple/Property)

incompatibility1 (label property) If their arguments satisfy some conditions, then two instantiations of *ei* are incompatible, i.e., no label may contain both.

$ei(arg_1)$ **incompatible with** $ei(arg_2)$ **if** $cond(arg_1, arg_2)$

pre-cond1 (transition property) If the label of a transition contains some instantiation of *ei*, then the source state of the transition must satisfy some condition.

if $ei(arg)$ **happen then** $cond(arg)$ where source state observers must appear in $cond(arg)$ and target state ones cannot appear

post-cond1 (transition property) If the label of a transition contains some instantiation of *ei*, then the target state of the transition must satisfy some condition). This may involve the source state.

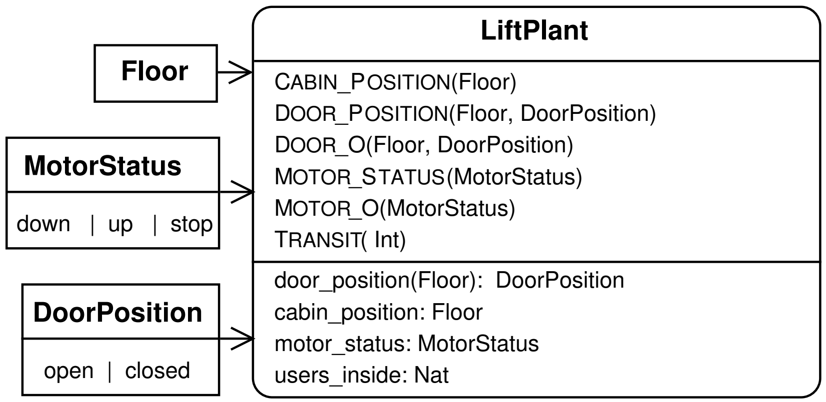
if $ei(arg)$ **happen then** $cond(arg)$ where target (primed) state observers must appear in $cond(arg)$ and source (non-primed) state ones may appear

Cell: About an elementary interaction (*ei*) - 2 (Simple/Property)

vitality1 (state property) If a state satisfies some condition, then any sequence of transitions starting from it will eventually contain a transition whose label contains *ei*. Note that vitality properties may have also the form “at least in a case” (instead of “in any case”) or “next” (instead of “eventually”).

if *cond*(*arg*) **then in any case eventually** *ei*(*arg*) **happen**

LiftPlant : Parts and Constituent Features (Simple/Property)



Parts: Floor, MotorStatus, DoorPosition

Constituent features

- *Elementary interactions*

CABIN_POSITION, DOOR_POSITION, DOOR_O, MOTOR_STATUS, MOTOR_O, TRANSIT

- *State observers*

door_position, *cabin_position*, *motor_status*, *users_inside*

Lift Plant properties - On MotorStatus (Simple/Property)

incompatibility1 (label property)

A sensor cannot signal two different values simultaneously.

MOTOR_STATUS(ms_1) **incompatible with** MOTOR_STATUS(ms_2) **if** $ms_1 \neq ms_2$

pre-cond1 (transition property)

A sensor always signals the correct data.

if MOTOR_STATUS(*ms*) **happen then** *motor_status* **=** *ms*

post-cond1 (transition property)

None

vitality1 (state property)

A sensor cannot break down, thus it may always be able to signal the correct value.

at least in one case next MOTOR_STATUS(*motor_status*) **happen**

Lift Plant properties - On the orders (Simple/Property)

Cell filling, drop repetition, rearrange, . . .

- Only appropriate groups of orders may be received simultaneously by the lift plant; precisely at most one order for the motor and one for the doors.

$\text{MOTOR\_O}(ms_1)$ **incompatible with** $\text{MOTOR\_O}(ms_2)$ **if** $ms_1 \neq ms_2$

$\text{DOOR\_O}(f_1, dps_1)$ **incompatible with** $\text{DOOR\_O}(f_2, dps_2)$ **if** . . .

- An order cannot be received when its execution may be problematic; precisely move up (down) only when the motor is stopped and the cabin is not at the top (ground) floor, and open the door at f only when no door is open, the cabin is at floor f and the motor is stopped.

if $\text{MOTOR\_O}(up)$ **happen then** $motor\_status = stop$ **and** $cabin\_position \neq top$

if $\text{MOTOR\_O}(down)$ **happen then** $motor\_status = stop$ **and** $cabin\_position \neq ground$

if $\text{DOOR\_O}(f_1, open)$ **happen then**

(**for all** f • **if** $f \neq f_1$ **then** $door\_position(f) \neq open$) **and** $cabin\_position = f_1$ **and** $motor\_status = stop$

- The orders are always correctly executed.

if $\text{MOTOR\_O}(ms)$ **happen then** $motor\_status' = ms$

if $\text{DOOR\_O}(f, dps)$ **happen then** $door\_position'(f) = dps$

CASL, CASL-LTL **view - (Simple/Property)**

- *poSpec*.parts = $\{$*ds*$_1$, ..., *ds*$_j\}$ data structure specifications

  DS$_1$, ..., DS$_j$ are the CASL-LTL presentations of *ds*$_1$, ..., *ds*$_j$
- *poSpec*.e-features $= \{ei_1, \ldots, ei_n\}$ the elementary interactions
- *poSpec*.s-features $= \{so_1, \ldots, so_m\}$ the state observers

spec ELINTERACTION =

free type *elInteraction* $::=$

ei$_1$.name(*ei*$_1$.argTypes) $|$ ... $|$ *ei*$_n$.name(*ei*$_n$.argTypes)

spec *poSpec*.name =

FINITESET[ELINTERACTION] **and** DS$_1$ **and** ... **and** DS$_j$ **then**

dsort *st* **label** *FinSet*[*elInteraction*]

ops *so*$_1$.name : *st* $\times$ *so*$_1$.argTypes $\to ?$ *so*$_1$.resType

...

so$_m$.name : *st* $\times$ *so*$_m$.argTypes $\to$ *so*$_m$.resType

axioms

formulae corresponding to the cell fillings

CASL CASL-LTL view: properties (Simple/Property)

- transition properties

	expressed by
cond	$S \xrightarrow{l} S' \Rightarrow$ *cond'*

where *cond'* is obtained from *cond* by adding

- S as extra argument to each source (non-primed) state observer,

- S' as extra argument to each target (primed) state observer,

and by the following replacement

	replaced by
"*eInt* **happen**"	*eInt* $\in l$

CASL CASL-LTL view: properties (follwd) (Simple/Property)

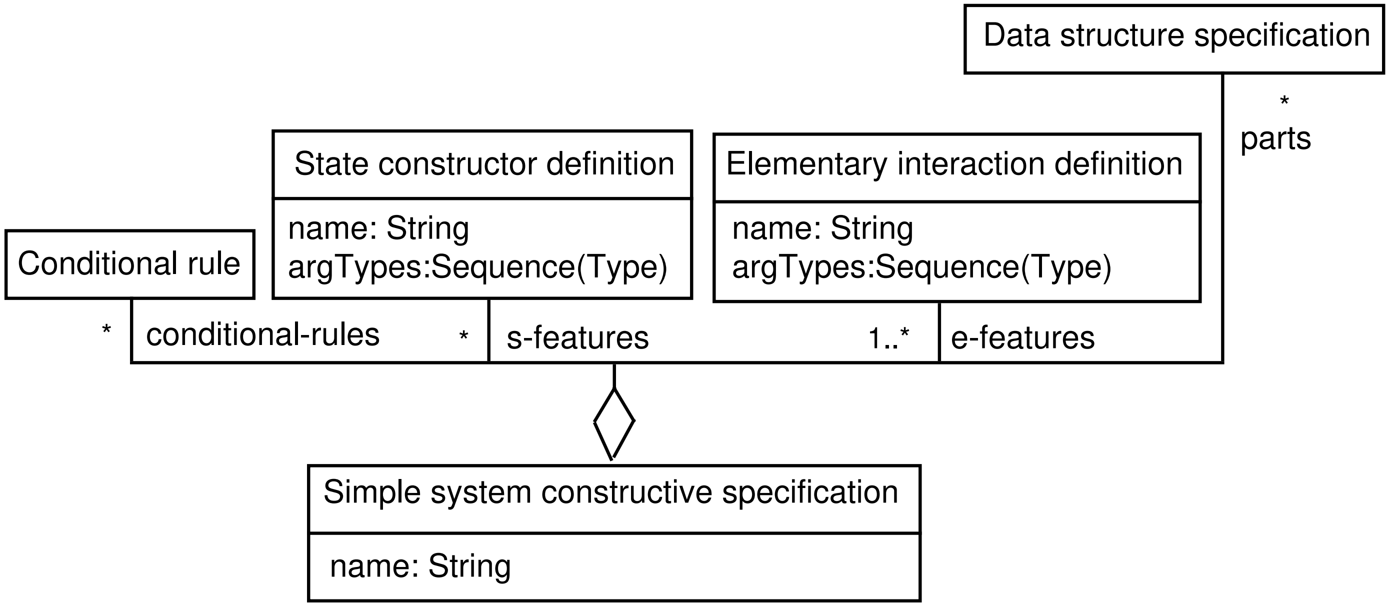
- label properties

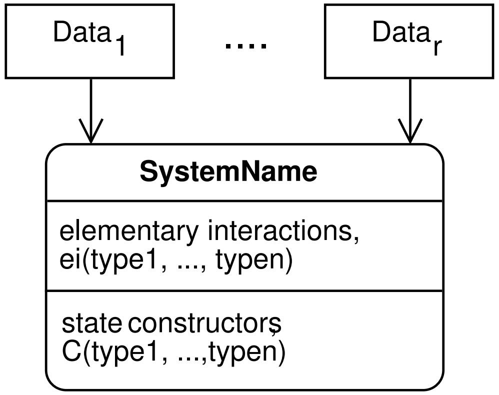
eInt1 **incompatible with** *eInt2* **if** *cond*	*cond* $\Rightarrow \neg$ (*eInt1* $\in l \wedge$ *eInt2* $\in l$) var l: $FinSet$[*elInteraction*]

- state properties

in any case ...	$in\_any\_case(S, \ldots)$
at least in one case ...	$in\_one\_case(S, \ldots)$
eventually *eInt*(arg) **happen**	$eventually\ < \ l \bullet$ *eInt*$(\mathrm{arg}) \in l\ >$
...	...

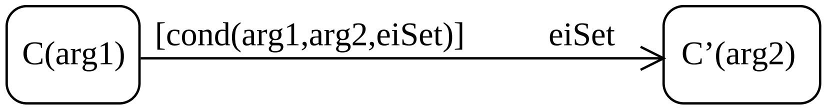
Constructive specifications (Simple systems)

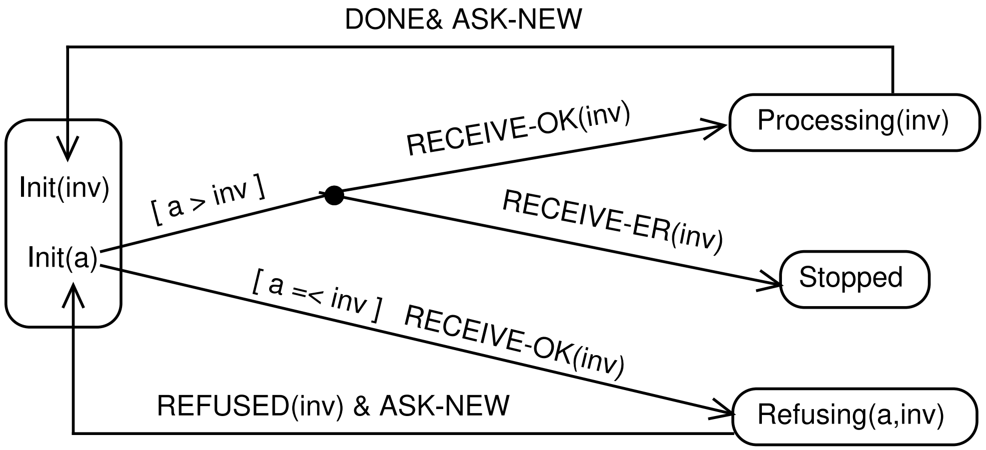




Towards a Formally Grounded Development Method

Constructive specifications (Simple systems) - Properties





if $a > inv$ **then** $Init(a) \xrightarrow{RECEIVE-OK(inv)} Processing(inv)$

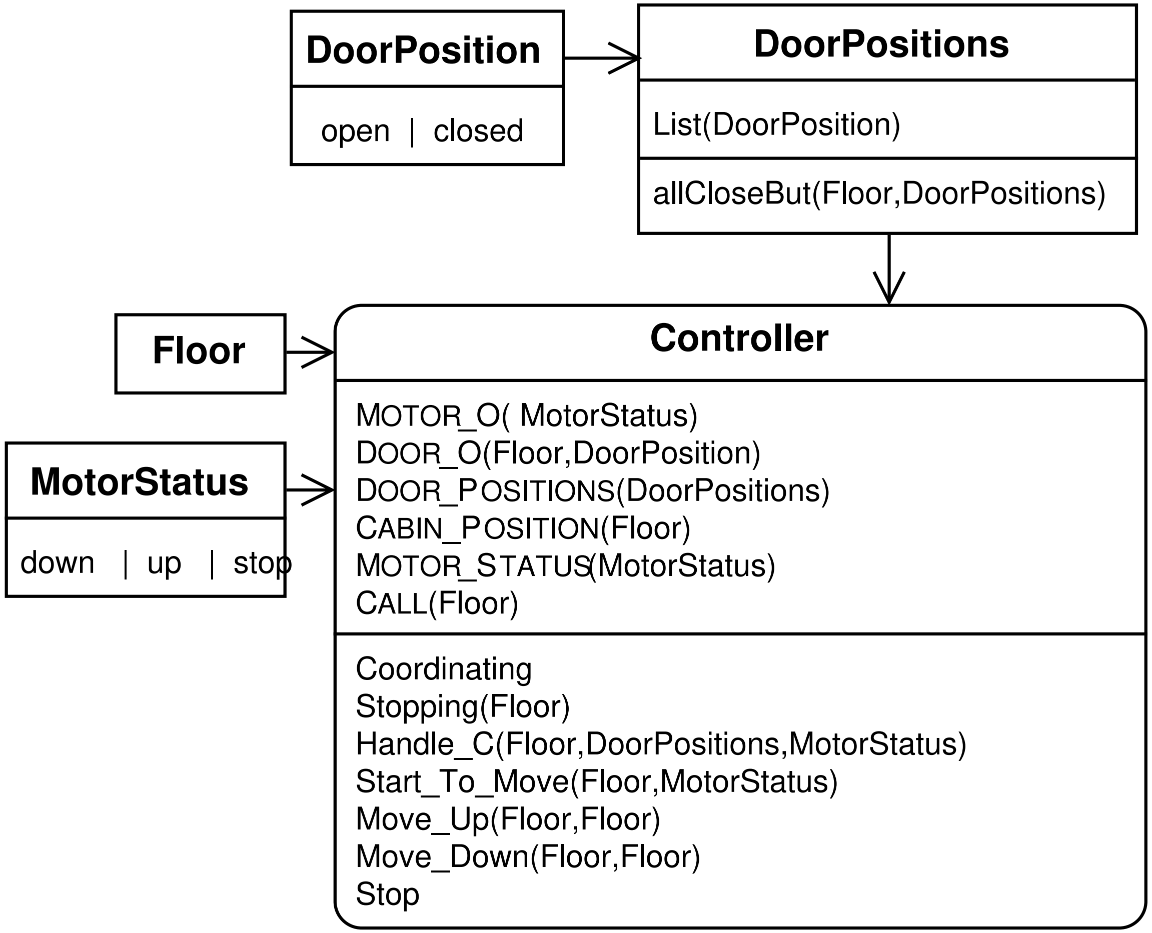
if $a > inv$ **then** $Init(a) \xrightarrow{RECEIVE-ER(inv)} Stopped$

if $a \leq inv$ **then** $Init(a) \xrightarrow{RECEIVE-OK(inv)} Refusing(a, inv)$

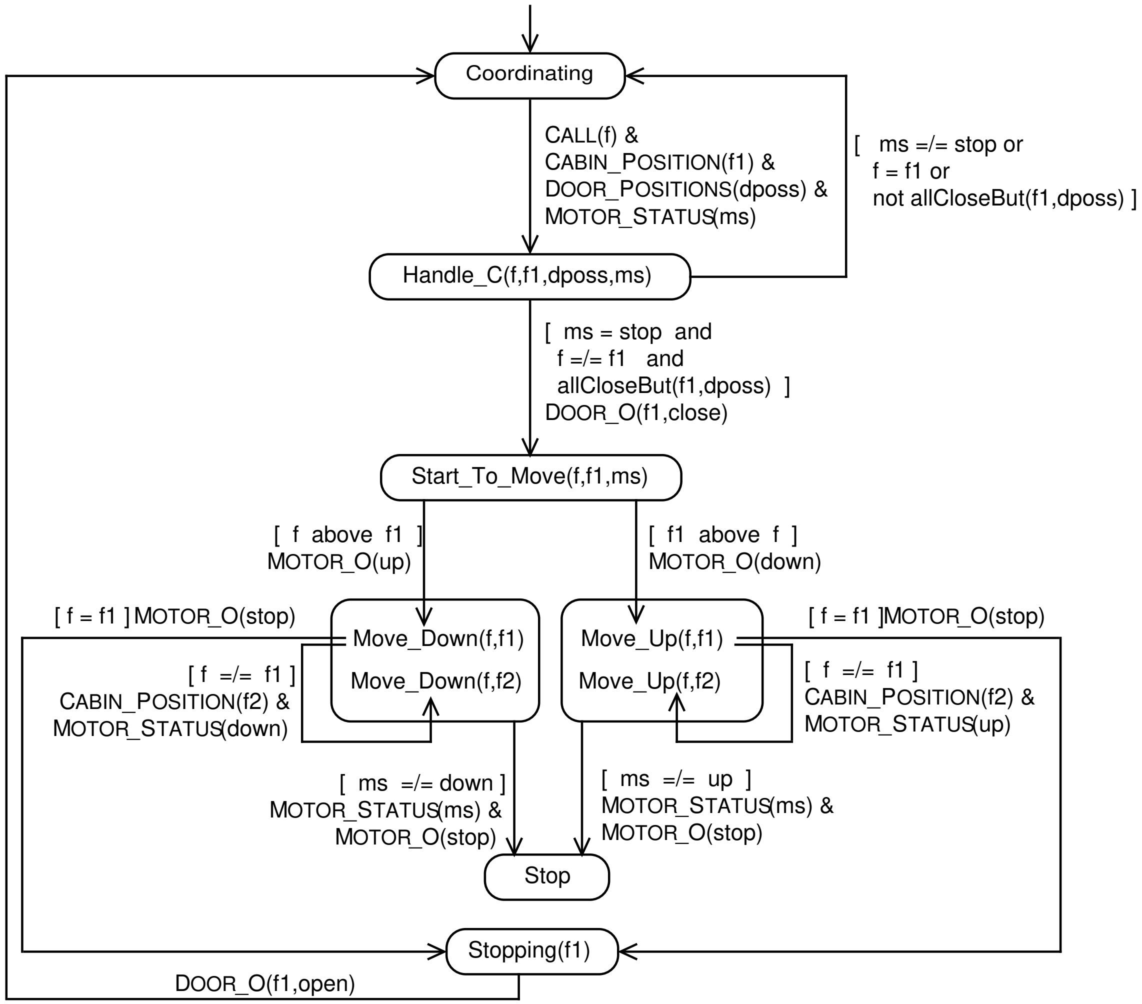
$$Refusing(a, inv) \xrightarrow{\{REFUSED(inv), ASK-NEW\}} Init(a)$$

$$Processing(inv) \xrightarrow{\{DONE, ASK-NEW\}} Init(inv)$$

Lift Controller (Simple/Constructive)



Lift Controller behaviour (Simple/Constructive)



CASL, CASL-LTL view: constructive spec of simple systems

- *conSpec*.parts = $\{ds_1, \ldots, ds_j\}$

  $DS_1, \ldots, DS_j$ are the CASL-LTL presentations of $ds_1, \ldots, ds_j$
- *conSpec*.e-features $= \{ei_1, \ldots, ei_n\}$ the elementary interactions
- *conSpec*.s-features $= \{sCon_1, \ldots, sCon_m\}$ the state constructors

spec ELINTERACTION =

free type *elInteraction* $::=$ ei_1.name(ei_1.argTypes) $| \ldots |$ ei_n.name(ei_n.argTypes)

spec *conSpec*.NAME =

FINITESET[ELINTERACTION] **and** DS_1 **and** ... **and** DS_j **then**

free {

dsort *st* **label** *FinSet*[*elInteraction*]

ops $sCon_1$.name : $sCon_1$.argTypes $\rightarrow$ *st*

...

$sCon_m$.name : *st* $\times$ $sCon_m$.argTypes $\rightarrow$ *st*

axioms

formulae corresponding to conditional rules

} **end**

Towards a Formally Grounded Development Method

Outline

Methods taking into account:

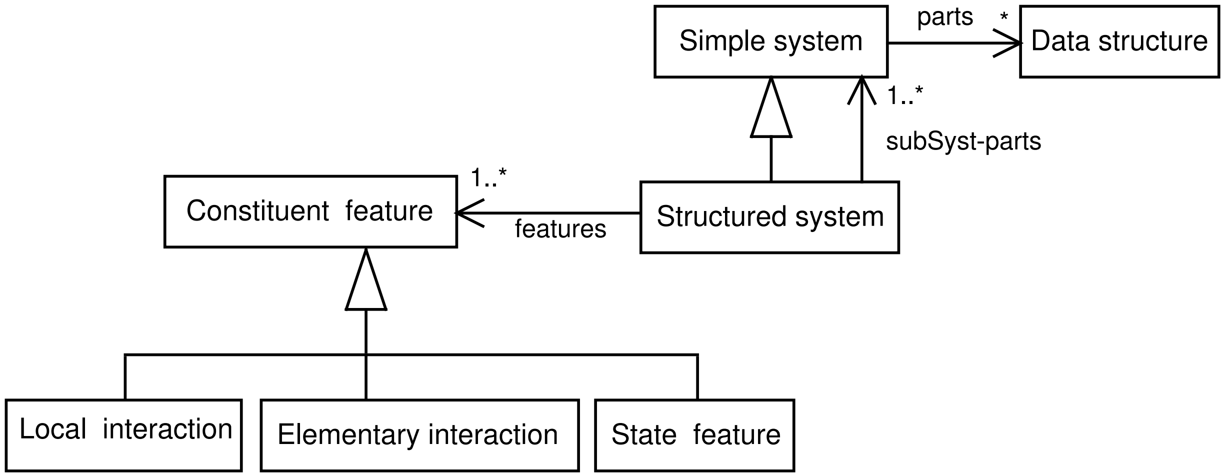
- a software item:
  - *a simple dynamic system*
  - a structured dynamic system
  - *a data structure*
- two specification techniques: *property-oriented, model-oriented* (constructive)
- CASL and CASL-LTL specifications

Illustration on case studies

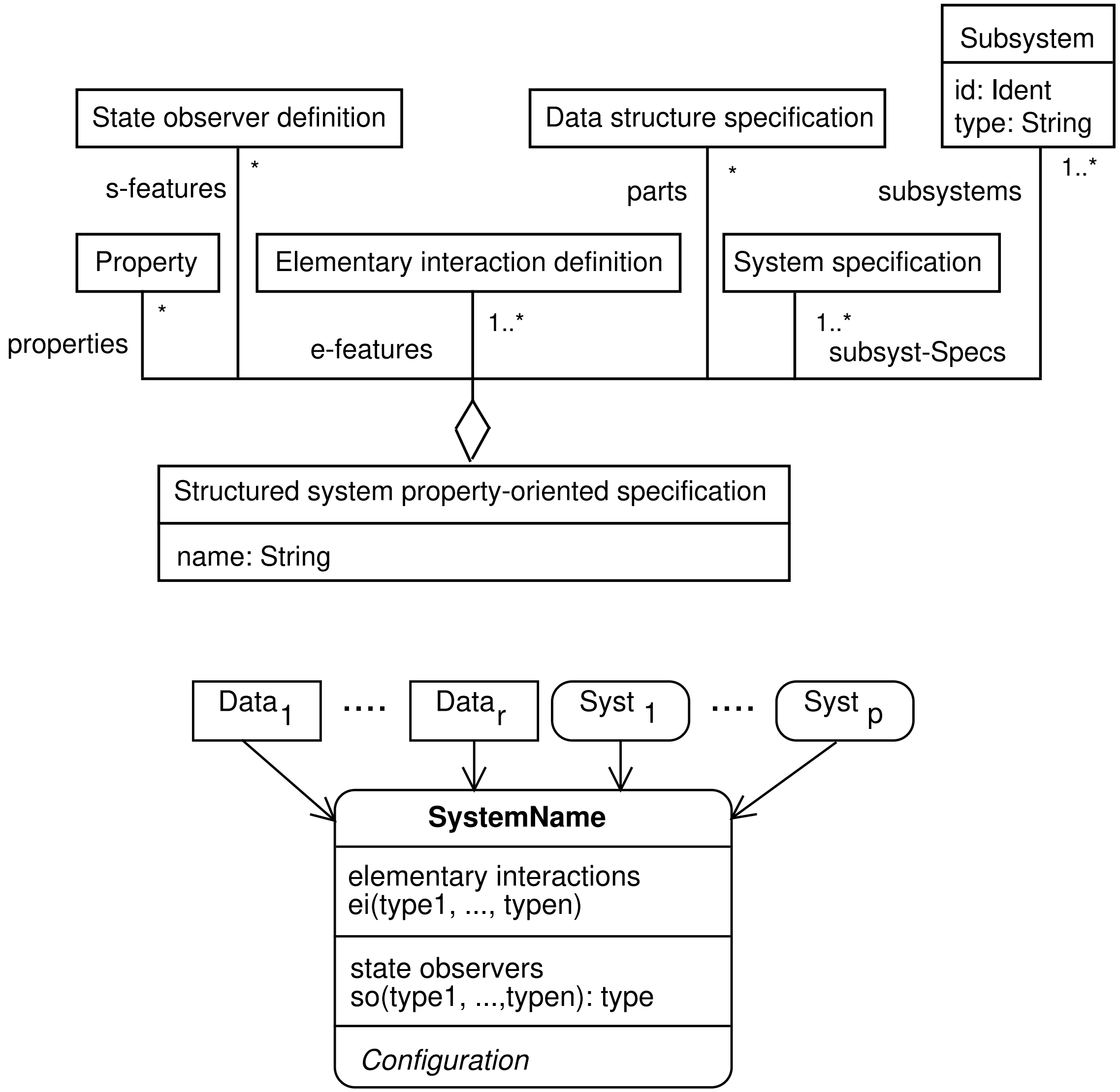
- in combination with structuring concepts as (Jackson's) problem frames

Structured Systems

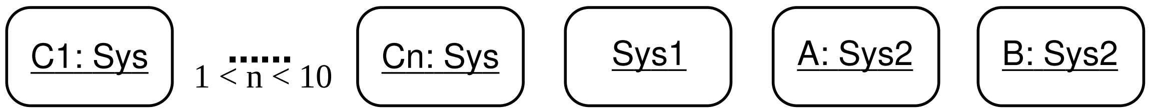
- specialization of the simple dynamic systems
- simple or structured *subsystems* uniquely identified by some identity
- situation: subsystems situations
- global move: simultaneous/concurrent executions of subsystems (local) moves
- *generalized lts* - information: set of local moves



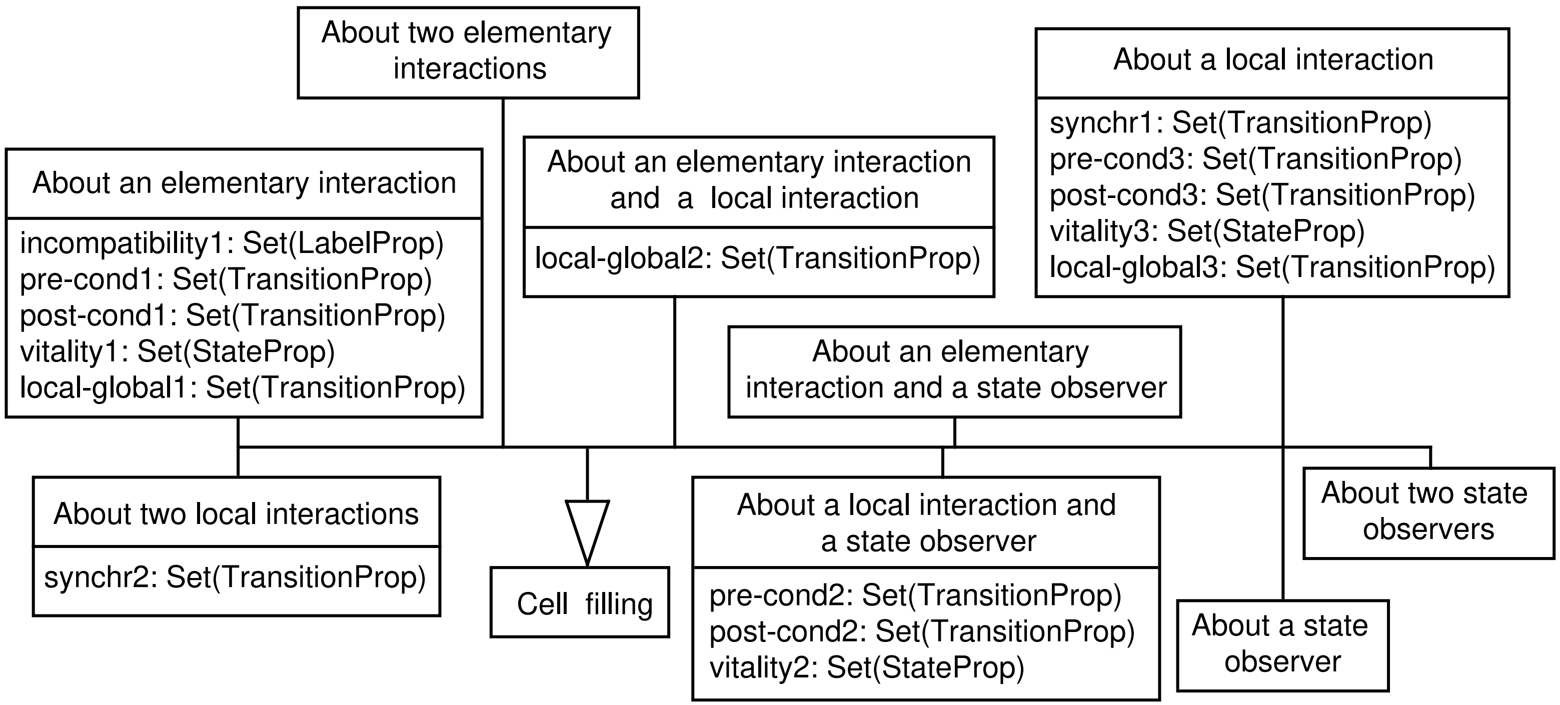
Property-oriented specifications of structured systems



Configuration and cells (Structured/Property)



Configuration



Cells

Cell example About a local interaction : synchr1 and local-global3 (Structured/Property)

synchr1 (transition property)

An instantiation of the local interaction is synchronized (i.e., executed simultaneously)/not synchronized with another instantiation of the same; clearly the two instantiations are performed by different subsystems.

if $cond(arg, arg_1)$ **and** $sid.ei(arg)$ **happen then** $sid_1.ei_1(arg_1)$ **happen**

or

if $cond(arg, arg_1)$ **and** $sid.ei(arg)$ **happen then not** $sid_1.ei_1(arg_1)$ **happen**

local-global3 (transition property)

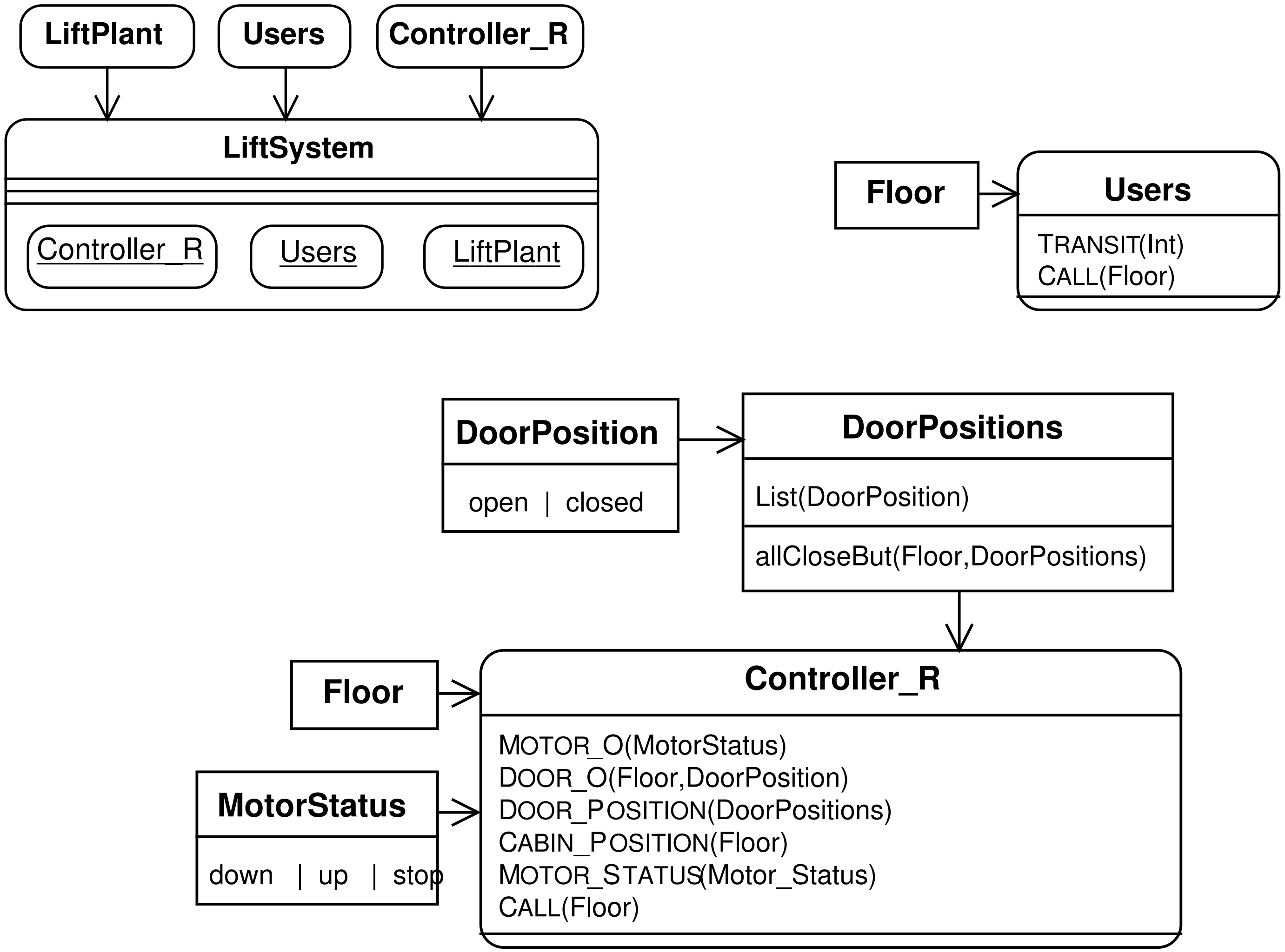
If an instantiation of *sid.ei* belongs to the label of some transition of some subsystem that is part of a global transition, then the label of such global transition must contain some elementary interaction, or vice versa.

if $sid.ei(arg)$ **happen and** $cond(arg, arg_1)$ **then** $ei_1(arg_1)$ **happen**

or

if $ei_1(arg_1)$ **happen and** $cond(arg, arg_1)$ **then** $sid.ei(arg)$ **happen**

Lift - Parts and Constituent Features (Structured/Property)



Lift Properties - (Structured/Property)

Local interactions with the same name and from different subsystems are synchronized

Users.CALL(f) **synchronized with** Controller_R.CALL(f)

LiftPlant.DOOR_POSITION(*ground*,dps_1), ... LiftPlant.DOOR_POSITION(*top*,dps_{10})
synchronized with Controller_R.DOOR_POSITIONS(dps_1 :: ... :: dps_{10})

if Users.CALL(f) **happen then in any case eventually**

LiftPlant.*cabin_position*(f) **and**

LiftPlant.*motor_status*(*stop*) **and**

LiftPlant.*door_position*(f) **=** *open*

CASL-LTL View (Structured/Property)

dsort *st* **label** *lab* **info** *inf* stands for **sorts** *st*, *lab*, *inf*
pred $\_\_ : \_\_ \xrightarrow{\_\_} \_\_ : inf \times st \times lab \times st$

- *poSpec*.parts = $\{ds_1, \ldots, ds_j\}$, and that $DS_1, \ldots, DS_j$ are the CASL-LTL presentations of the data structure specifications $ds_1, \ldots, ds_j$ respectively
- *poSpec*.subsyst-Specs $= \{ssp_1, \ldots, ssp_k\}$, that $SSP_1, \ldots, SSP_k$ are the CASL-LTL presentations of the system specifications $ssp_1, \ldots, ssp_k$ respectively, and that $\text{ELINTERACTION}_1, \ldots, \text{ELINTERACTION}_k$ be the specifications of their elementary interactions.
- *poSpec*.e-features $= \{ei_1, \ldots, ei_n\}$ the elementary interactions
- *poSpec*.s-features $= \{so_1, \ldots, so_m\}$ the state observers
- *poSpec*.subsystems $= \{ss_1, \ldots, ss_r\}$ the subsystems

CASL-LTL View foll'd (Structured/Property)

spec LOCALINTERACTION =

ELINTERACTION$_1$ **and** ... **and** ELINTERACTION$_k$ **and** Ident **then**

free type *subElInteraction* $::=$ $\_(elInteraction_1) \mid \ldots \mid \_(elInteraction_k)$

%% disjoint union of the elementary interaction types of the subsystems

free type *localInteraction* $::=$ $< \_, \_ >$ $($*ident*, *subElInteraction*$)$

spec *poSpec*.name =

FINITESET[ELINTERACTION] **and** FINITESET[LOCALINTERACTION] **and**

DS$_1$ **and** ... **and** DS$_j$ **and** SSP$_1$ **and** ... **and** SSP$_k$ **then**

dsort *st* **label** *FinSet*[*elInteraction*] **info** *FinSet*[*localInteraction*]

ops so_1.name : *st* $\times$ so_1.argTypes $\rightarrow$ so_1.resType **%%** state observers

...

so_m.name : *st* $\times$ so_m.argTypes $\rightarrow$ so_m.resType

ss_1.id : *st* $\rightarrow$ ss_1.type **%%** observers of the subsystem states

...

ss_r.id : *st* $\rightarrow$ ss_r.type

axioms *those formulae corresponding to the cell fillings*

Outline

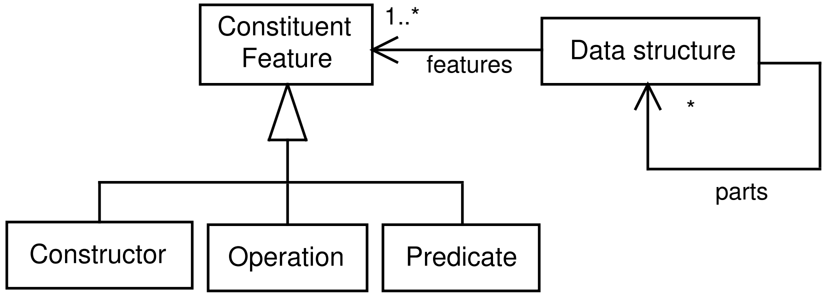
Methods taking into account:

- a software item:
  - *a simple dynamic system*
  - *a structured dynamic system*
  - a data structure
- two specification techniques: *property-oriented, model-oriented* (constructive)
- CASL and CASL-LTL specifications

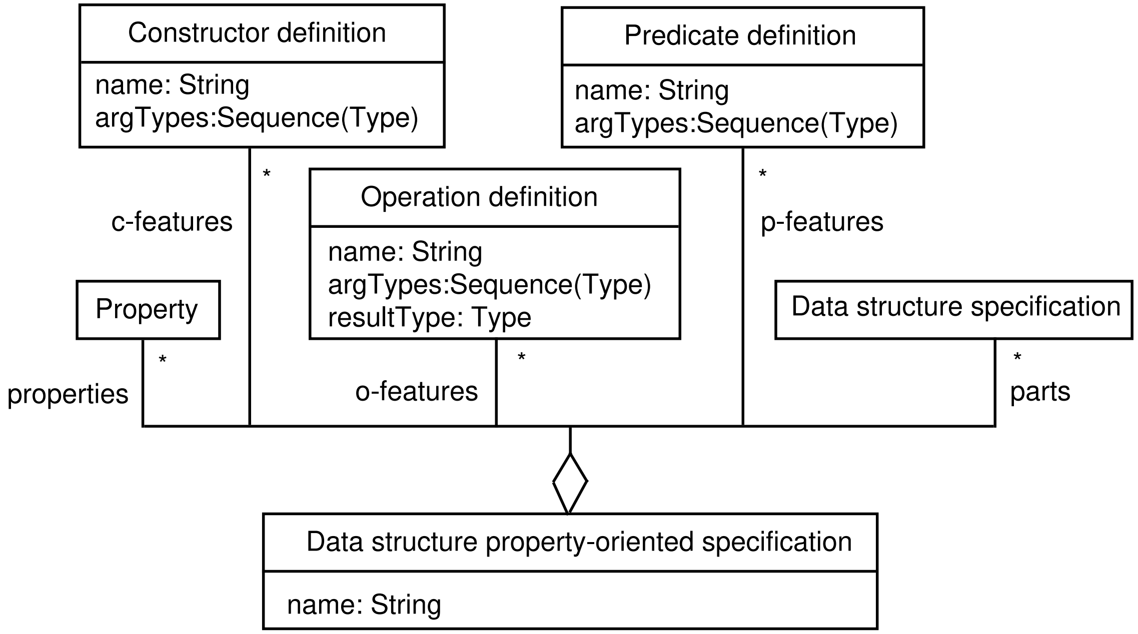
Illustration on case studies

- in combination with structuring concepts as (Jackson's) problem frames

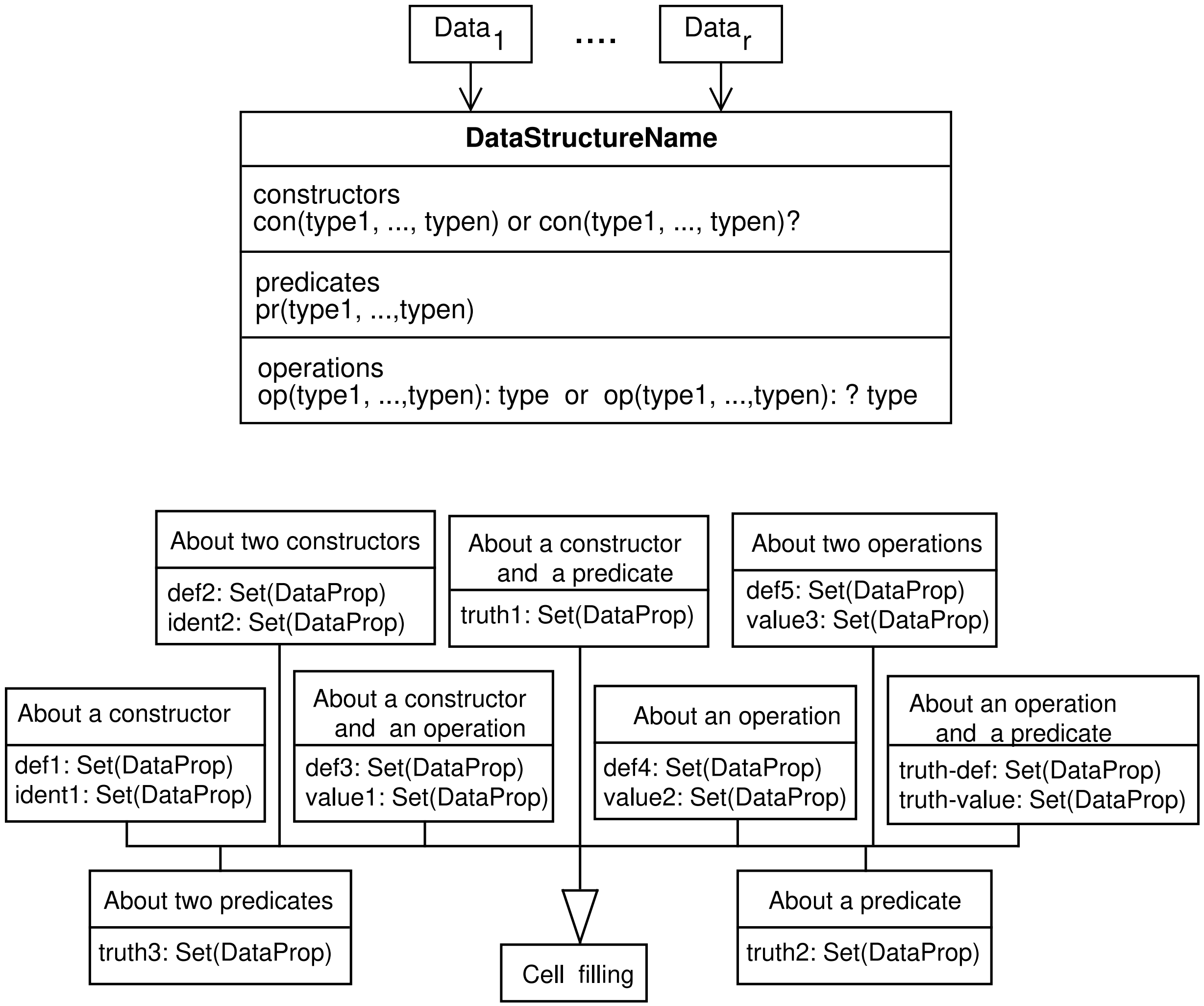
Data Structure Items / Property



Property-oriented



Data Structure - Property-oriented



Floor (Data Structure/Property-oriented)

Floor
ground top
_ above _(Floor,Floor)
next(Floor): ? Floor previous(Floor): ? Floor

– There exists a ground and a top floor, and they are different.

$ground \neq top$

– *next* returns the floor immediately above a given one, if it exists.
There is no floor between f and $next(f)$.

def($next(ground)$)

not def($next(top)$)

def($next(f)$) **iff** top $above$ f

whenever everything is defined

$next(f)$ $above$ f **and not exists** f_1 $\bullet$ ($next(f)$ $above$ f_1 **and** f_1 $above$ f)

whenever everything is defined $next(previous(f)) = previous(next(f)) = f$

– *above* is total order over the floors with *top* as maximum and *ground* as minimum . . . .

CASL View (Data/Property)

- *poSpec*.parts = $\{$*ds*$_1$, ..., *ds*$_j\}$ w/ DS$_1$, ..., DS$_j$ CASL-LTL presentations
- *poSpec*.c-features = $\{$*con*$_1$, ..., *con*$_n\}$ the constructors
- *poSpec*.o-features = $\{$*op*$_1$, ..., *op*$_m\}$ the operations
- *poSpec*.p-features = $\{$*pr*$_1$, ..., *pr*$_p\}$ the predicates.

spec *poSpec*.name =

DS$_1$ **and** ... **and** DS$_j$ **then**

type *poSpec*.name $::=$ con_1.name$($$con_1$.argTypes$)?$ $|$ ... $|$ con_n.name$($$con_n$.argTypes$)$

ops *op*$_1$.name : *op*$_1$.argTypes $\rightarrow ?$ *op*$_1$.resType

...

op$_m$.name : *op*$_m$.argTypes $\rightarrow$ *op*$_m$.resType

preds *pr*$_1$.name : *pr*$_1$.argTypes

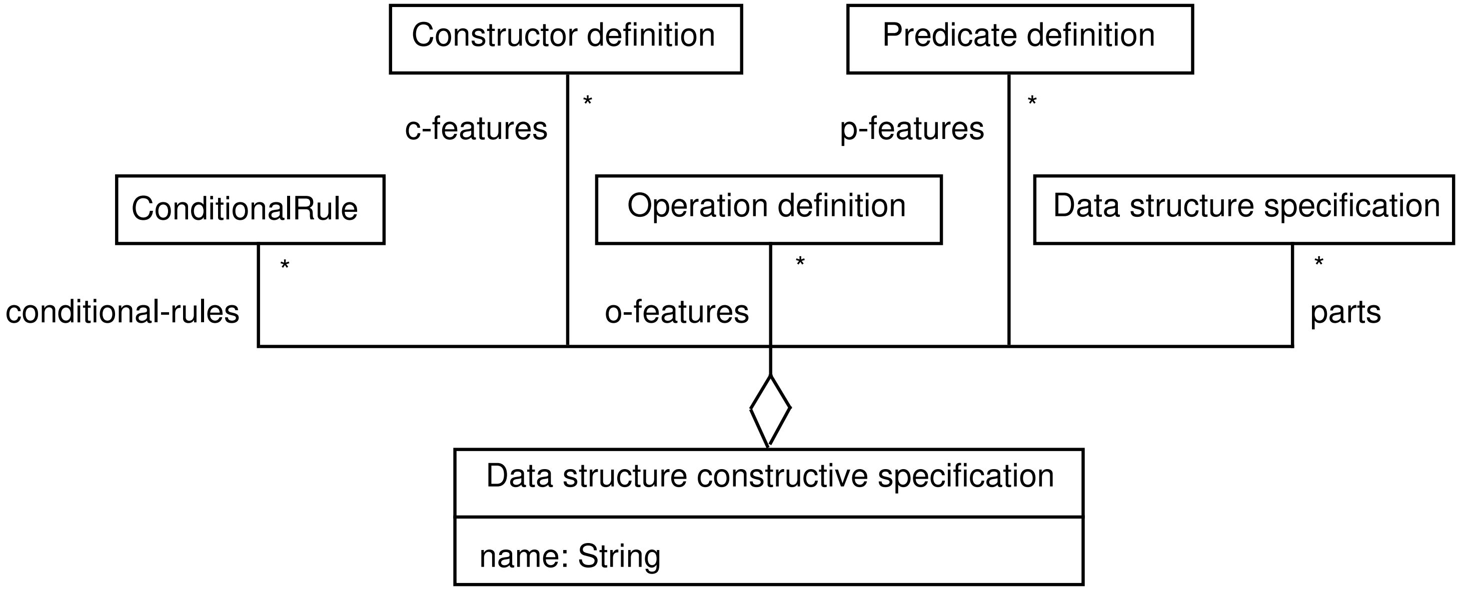
...

pr$_p$.name : *pr*$_p$.argTypes

axioms

formulae corresponding to the cell fillings

Data Structure - Constructive



Towards a Formally Grounded Development Method

Outline

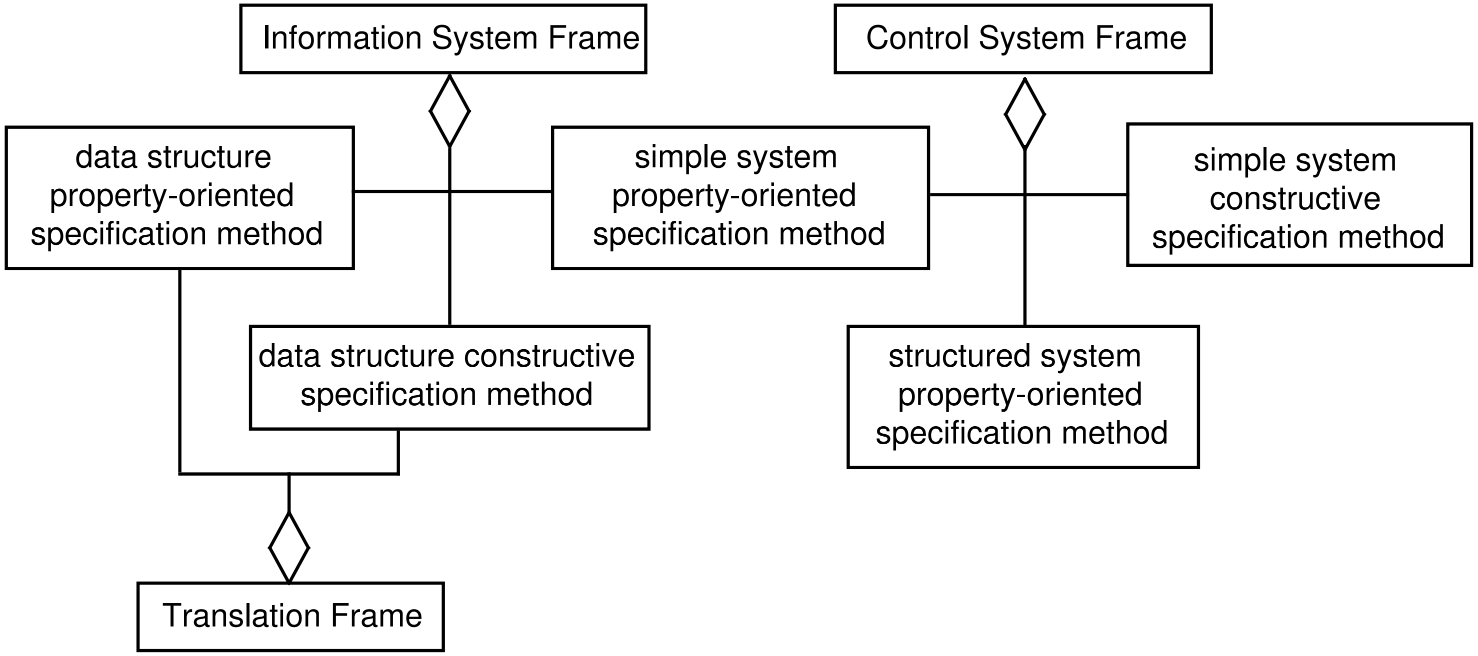
Methods taking into account:

- a software item:
  - *a simple dynamic system*
  - *a structured dynamic system*
  - *a data structure*
- two specification techniques: *property-oriented, model-oriented* (constructive)
- CASL and CASL-LTL specifications

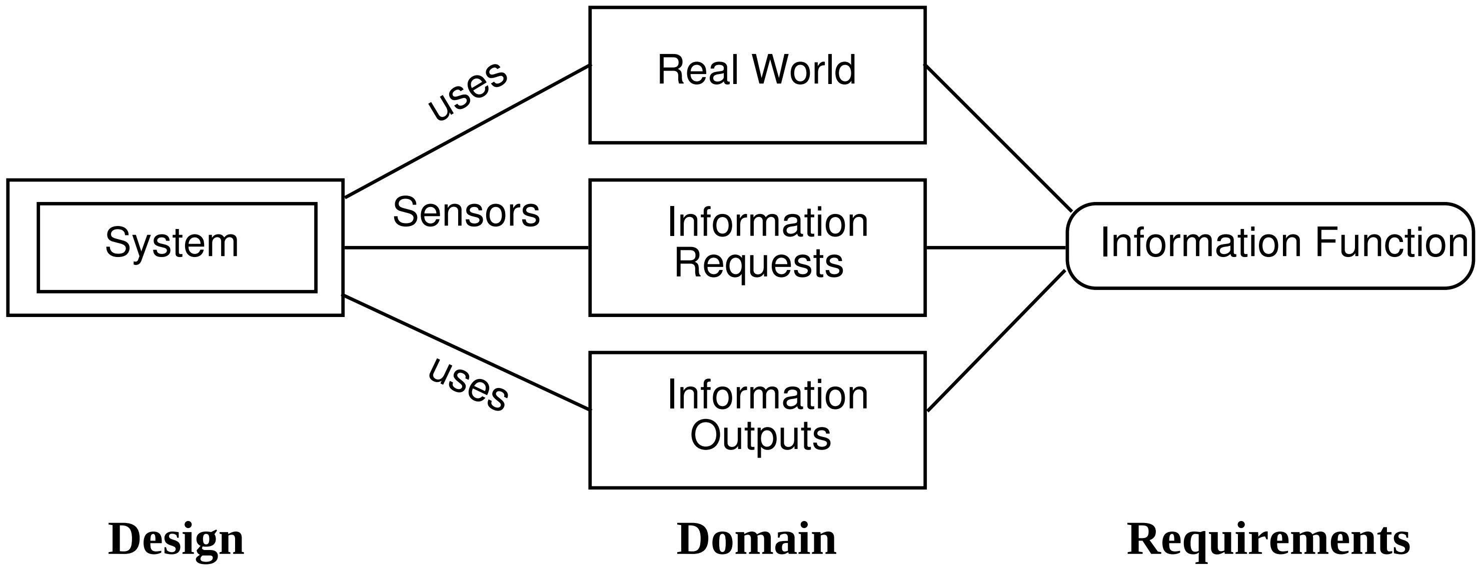
Illustration on case studies

- in combination with structuring concepts as (Jackson's) problem frames

Applying our Specification Methods to Classes of Systems ("Problem Frames")



Information System Frame



Real World : simple dynamic system (property), signals relevant information

Information Requests/Outputs : data structure (model/constructive)

Information function : with a (model/constructive) data structure (History, ...)

System : simple system (model/constructive)

Conclusion and . . .

Companion method for (algebraic) formal specifications

- Paradigms, techniques, pragmatic characteristics originated from the underlying theory (e.g. no “use cases” . . . , no OO)
- both visual and explicit presentations
- systematic and inherently rigorous, cell-filling
- well defined underlying formal models
- experimented on sizeable case studies, on students
- “building-bricks” specification tasks for different kinds of software (simple systems, structured, data structures), at different abstraction level (property/more abstract, model or constructive/more concrete)
- relevant for real applications, used for requirement specifications, or in connection with structuring concepts (problem frames)

. . . Perspectives

- Our cell-filling technique can be a basis for generating precise UML models, or for their inspection (checking all aspects considered)
- Further experiments, new problem frames (business automation, web applications, distributed mobile systems, . . .)
- Oriented towards CASL and CASL-LTL (algebraic specifications) but adaptable to other specification/description paradigms
- Supporting tools (graphical editor, type checker, guidelines support, . . .)