Formally Grounded Specifications
Requirements Methodology

C. Choppy
LIPN, Université Paris XIII France
http://lipn.univ-paris13.fr/~choppy

Joint work with

G. Reggio
DISI, Università di Genova, Italy
Plan

• A formally grounded development method for formal requirements specifications
• Link with “less formal” use cases
• …
### 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 !?

<table>
<thead>
<tr>
<th></th>
<th>FORMAL</th>
<th>INFORMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>notation</td>
<td>not very friendly (exotic)</td>
<td>very friendly, visual</td>
</tr>
<tr>
<td>notation</td>
<td>rigid, overhead</td>
<td>flexible, adaptable</td>
</tr>
<tr>
<td>learning</td>
<td>time, background</td>
<td>short(?)</td>
</tr>
<tr>
<td>case studies</td>
<td>simple (?)</td>
<td>real common app</td>
</tr>
</tbody>
</table>

Towards a Formally Grounded Development Method
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
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
• **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.

Towards a Formally Grounded Development Method
**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 \(< sort \ st, \ lab \>

\[ \text{pred } - \xrightarrow{\_} - : st \times lab \times st \]

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 . . . .

Towards a Formally Grounded Development Method
A General Property-oriented Specification Method (GPSm)

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

Towards a Formally Grounded Development Method

Jan 2005
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 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)

State observer definition
- name: String
- argTypes: Sequence(Type)
- resultType: Type

Elementary interaction definition
- name: String
- argTypes: Sequence(Type)

Data structure specification

Simple system property-oriented specification
- name: String

Visual presentation
- Data₁, ..., Dataᵣ
- SystemName
  - elementary interactions, E₁(type₁, ..., typen)
  - state observers, so₁(type₁, ..., typen): type

Cell filling
- Elementary Interaction
- State Observer
  - E₁
  - so₁
  - Ele₁, so₁
  - E₁, so₁

Towards a Formally Grounded Development Method

Jan 2005
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) = v₁ and so'(arg) = v₂ and v₁ ≠ v₂ then cond(v₁,v₂,arg) and ei₁,...,eiₙ happen

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

if cond(v₁,v₂,arg) and so(arg) = v₁ and v₁ ≠ v₂ then in any case eventually so(arg) = v₂

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.

\[ ei(arg_1) \text{ incompatible with } ei(arg_2) \text{ if } cond(arg_1, arg_2) \]

**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 \( ei(arg) \text{ happen then } cond(arg) \) 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 \( ei(arg) \text{ happen then } cond(arg) \) 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”).

\[
\text{if } \text{cond}(\text{arg}) \text{ then in any case eventually } ei(\text{arg}) \text{ happen}
\]
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

Towards a Formally Grounded Development Method

Jan 2005

IFIP WG1.3
incompatibility1 (label property)
A sensor cannot signal two different values simultaneously.

\[ \text{MOTOR\_STATUS}(ms_1) \text{ incompatible with MOTOR\_STATUS}(ms_2) \text{ if } ms_1 \neq ms_2 \]

pre-cond1 (transition property)
A sensor always signals the correct data.

\[ \text{if MOTOR\_STATUS}(ms) \text{ happen then } motor\_status = ms \]

post-cond1 (transition property)
None

vitality1 (state property)
A sensor cannot break down, thus it may always be able to signal the correct value.

\[ \text{at least in one case next MOTOR\_STATUS}(motor\_status) \text{ happen} \]
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.

\[
\text{\textit{MOTOR}}_O(ms_1) \text{ incompatible with} \text{\textit{MOTOR}}_O(ms_2) \text{ if } ms_1 \neq ms_2
\]

\[
\text{\textit{DOOR}}_O(f_1,dps_1) \text{ incompatible with} \text{\textit{DOOR}}_O(f_2,dps_2) \text{ if } ...
\]

- 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.

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

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

\[
\text{if } \text{\textit{DOOR}}_O(f,\text{open}) \text{ happen then}
\]

\[
\text{(for all } f \text{ • if } f \neq f_1 \text{ then } door\_position(f) \neq \text{open) and } cabin\_position = f_1 \text{ and } motor\_status = \text{stop}}
\]

- The orders are always correctly executed.

\[
\text{if } \text{\textit{MOTOR}}_O(ms) \text{ happen then } motor\_status' = ms
\]

\[
\text{if } \text{\textit{DOOR}}_O(f,dps) \text{ happen then } door\_position'(f) = dps
\]

Towards a Formally Grounded Development Method
**Casl, Casl-Ltl View - (Simple/Property)**

- `pospec.parts = \{ds_1, \ldots, ds_j\}` data structure specifications
  - `DS_1, \ldots, DS_j` are the Casl-Ltl presentations of `ds_1, \ldots, ds_j`
- `pospec.e-features = \{ei_1, \ldots, ei_n\}` the elementary interactions
- `pospec.s-features = \{so_1, \ldots, so_m\}` the state observers

```plaintext
spec ELInteraction =
  free type ellInteraction ::=     
      ei_1.name(ei_1.argTypes) | \ldots | ei_n.name(ei_n.argTypes)

spec pospec.name =
  FINITESET[ELInteraction] and DS_1 and \ldots and DS_j then
  dsort st label FinSet[ellInteraction]
  ops so_1.name : st \times so_1.argTypes \rightarrow so_1.resType
  \ldots
      so_m.name : st \times so_m.argTypes \rightarrow so_m.resType

axioms
  formulae corresponding to the cell fillings
```

Towards a Formally Grounded Development Method
**transition properties**

<table>
<thead>
<tr>
<th>cond</th>
<th>$S \xrightarrow{l} S'$ ⇒ cond'</th>
</tr>
</thead>
</table>

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

| “elnt happen” | replaced by elnt ∈ l |
**CASL-CASL-LTL**  
**view: properties (follwd) (Simple/Property)**

- **label properties**

  $$eInt_1 \text{ incompatible with } eInt_2 \text{ if } cond$$

  $$cond \Rightarrow \neg (eInt_1 \in l \land eInt_2 \in l)$$

  $$\text{var } l: FinSet[\text{elInteraction}]$$

- **state properties**

  **in any case** ...

  **at least in one case** ...

  **eventually** $eInt(\text{arg})$ **happen**

  **...**

  $$\text{in\_any\_case}(S, \ldots)$$

  $$\text{in\_one\_case}(S, \ldots)$$

  $$\text{eventually } < l \bullet eInt(\text{arg}) \in l >$$

  **...**

---

Towards a Formally Grounded Development Method

Jan 2005
Constructive specifications (Simple systems)

State constructor definition
- name: String
- argTypes: Sequence(Type)

Elementary interaction definition
- name: String
- argTypes: Sequence(Type)

Simple system constructive specification
- name: String

Data structure specification
- parts

Conditional rule
- * conditional-rules
- * s-features
- 1..* e-features

Data_1 .... Data_r

SystemName
- elementary interactions, ei(type1, ..., typen)
- state constructors C(type1, ..., typen)

Towards a Formally Grounded Development Method

Jan 2005
Constructive specifications (Simple systems) - Properties

\[
\begin{align*}
C(\arg_1) & \quad \text{[cond}(\arg_1, \arg_2, \text{eiSet})] \quad \text{eiSet} \quad \Rightarrow \quad C'(\arg_2) \\
\end{align*}
\]

**Processes**

- **Init(inv)**
- **Init(a)**
- **Processing(inv)**
- **Stopped**
- **Refusing(a, inv)**
- **Refused(inv)** & **ASK-NEW**

**Transitions**

- **If** \( a > \text{inv} \) **then** **Init**(a) \( \xrightarrow{\text{RECEIVE-OK}(\text{inv})} \) **Processing**(inv)
- **If** \( a > \text{inv} \) **then** **Init**(a) \( \xrightarrow{\text{RECEIVE-ER}(\text{inv})} \) **Stopped**
- **If** \( a \leq \text{inv} \) **then** **Init**(a) \( \xrightarrow{\text{RECEIVE-OK}(\text{inv})} \) **Refusing**(a, inv)
- **Refusing**(a, inv) \( \xrightarrow{\{\text{REFUSED}(\text{inv}), \text{ASK-NEW}\}} \) **Init**(a)
- **Processing**(inv) \( \xrightarrow{\{\text{DONE, ASK-NEW}\}} \) **Init**(inv)

Towards a Formally Grounded Development Method

Jan 2005
Lift Controller (Simple/Constructive)

### Controller
- MOTOR_O(MotorStatus)
- DOOR_O(Floor,DoorPosition)
- DOORPOSITIONS(DoorPositions)
- CABIN_POSITION(Floor)
- MOTOR_STATUS(MotorStatus)
- CALL(Floor)

#### Coordinating
- Stopping(Floor)
- Handle_C(Floor,DoorPositions,MotorStatus)
- Start_To_Move(Floor,MotorStatus)
- Move_Up(Floor,Floor)
- Move_Down(Floor,Floor)
- Stop

#### DoorPosition
- open | closed

#### DoorPositions
- List(DoorPosition)
- allCloseBut(Floor,DoorPositions)

#### Floor

#### MotorStatus
- down | up | stop

Towards a Formally Grounded Development Method

Jan 2005
Lift Controller behaviour (Simple/Constructive)

Coordinating

CALL(f) &
CABIN_POSITION(f1) &
DOOR_POSITIONS(dposs) &
MOTOR_STATUS(ms)

[ ms /= stop or
f = f1 or
not allCloseBut(f1,dposs) ]

Handle_C(f1,dposs,ms)

[ ms = stop and
f /= f1 and
allCloseBut(f1,dposs) ]

DOOR_O(f1,close)

Start_To_Move(f1,ms)

[ f above f1 ]

MOTOR_O(up)

Move_Down(f1)

[ f = f1 ]

MOTOR_O(stop)

Move_Down(f2)

[ f /= f1 ]

CABIN_POSITION(f2) &
MOTOR_STATUS(down)

[ ms /= down ]

MOTOR_STATUS(ms) &
MOTOR_O(stop)

Stop

Towards a Formally Grounded Development Method

Jan 2005

IFIP WG1.3
**CASL, CASL-LTL** view: constructive spec of simple systems

- \(\text{conSpec.parts} = \{ds_1, \ldots, ds_j\}\)
  
  \(DS_1, \ldots, DS_j\) are the CASL-LTL presentations of \(ds_1, \ldots, ds_j\)

- \(\text{conSpec.e-features} = \{ei_1, \ldots, ei_n\}\) the elementary interactions

- \(\text{conSpec.s-features} = \{sCon_1, \ldots, sCon_m\}\) the state constructors

```plaintext
spec ELInteraction =
  free type ellInteraction ::= ei_1.name(ei_1.argTypes) \mid \ldots \mid ei_n.name(ei_n.argTypes)

spec conSpec.NAME =
  \text{FINITESET}[\text{ELInteraction}] \text{ and } DS_1 \text{ and } \ldots \text{ and } DS_j \text{ then}

free {
  dsort st label FinSet[ellInteraction]
  ops sCon_1.name : sCon_1.argTypes \rightarrow st
  \ldots

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

  axioms
  
  \text{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

Towards a Formally Grounded Development Method
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

**Towards a Formally Grounded Development Method**

Jan 2005
Transition of a structured system

- Local elementary interactions: A.e  A.f ...
- Global elementary interactions: X Y
Property-oriented specifications of structured systems

- State observer definition
- Data structure specification
- Elementary interaction definition
- Subsystem
  - id: Ident
  - type: String
- System specification
  - parts
  - subsystems
- Structured system property-oriented specification
  - name: String

Data 1, ..., Data_r, Syst_1, ..., Syst_p

SystemName

- elementary interactions ei(type1, ..., typen)
- state observers so(type1, ..., typen): type

Towards a Formally Grounded Development Method
**Configuration and cells (Structured/Property)**

**Configuration**

- **C1: Sys**
- **Cn: Sys**
- **Sys1**
- **A: Sys2**
- **B: Sys2**

- **1 < n < 10**

**About two elementary interactions**

- **About an elementary interaction**
  - incompatibility1: Set(LabelProp)
  - pre-cond1: Set(TransitionProp)
  - post-cond1: Set(TransitionProp)
  - vitality1: Set(StateProp)
  - local-global1: Set(TransitionProp)

- **About two local interactions**
  - synchr2: Set(TransitionProp)

- **Cell filling**

**About a local interaction**

- **About an elementary interaction and a local interaction**
  - synchr1: Set(TransitionProp)
  - pre-cond3: Set(TransitionProp)
  - post-cond3: Set(TransitionProp)
  - vitality3: Set(StateProp)
  - local-global3: Set(TransitionProp)

- **About an elementary interaction and a state observer**

- **Cell filling**

- **About a local interaction and a state observer**
  - pre-cond2: Set(TransitionProp)
  - post-cond2: Set(TransitionProp)
  - vitality2: Set(StateProp)

- **About two state observers**

**Cells**

**Towards a Formally Grounded Development Method**

Jan 2005  IFIP WG1.3
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.

\[
\text{if } \text{cond}(arg, arg_1) \text{ and } \text{sid.ei}(arg) \text{ happen then } \text{sid}_1.\text{ei}_1(arg_1) \text{ happen}
\]
or

\[
\text{if } \text{cond}(arg, arg_1) \text{ and } \text{sid.ei}(arg) \text{ happen then not } \text{sid}_1.\text{ei}_1(arg_1) \text{ happen}
\]

**local-global3** (transition property)

If an instantiation of \textit{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.

\[
\text{if } \text{sid.ei}(arg) \text{ happen and } \text{cond}(arg, arg_1) \text{ then } \text{ei}_1(arg_1) \text{ happen}
\]
or

\[
\text{if } \text{ei}_1(arg_1) \text{ happen and } \text{cond}(arg, arg_1) \text{ then } \text{sid.ei}(arg) \text{ happen}
\]
Lift - Parts and Constituent Features (Structured/Property)

- LiftSystem
  - LiftPlant
  - Users
  - Controller_R

- Floor
  - Users
    - TRANSIT(Int)
    - CALL(Floor)

- DoorPosition
  - open
  - closed

- DoorPositions
  - List(DoorPosition)
  - allCloseBut(Floor, DoorPositions)

- Controller_R
  - MOTOR_O(MotorStatus)
  - DOOR_O(Floor, DoorPosition)
  - DOOR_POSITION(DoorPositions)
  - CABIN_POSITION(Floor)
  - MOTOR_STATUS(Motor_Status)
  - CALL(Floor)

Towards a Formally Grounded Development Method

Jan 2005
Local interactions with the same name and from different subsystems are synchronized

Users.CALL(f)  \textbf{synchronized with} Controller.R.CALL(f)

LiftPlant.DOOR\_POSITION(\textit{ground},dps_1), \ldots LiftPlant.DOOR\_POSITION(\textit{top},dps_{10}) \textbf{synchronized with} Controller.R.DOOR\_POSITIONS(dps_1::\ldots::dps_{10})

\begin{verbatim}
if Users.CALL(f)  \textbf{happen then in any case eventually}
    LiftPlant.cabin\_position(f)  \textbf{and}  
    LiftPlant.motor\_status(stop)  \textbf{and}  
    LiftPlant.door\_position(f) = open
\end{verbatim}
**CASL-LTL View (Structured/Property)**

\begin{itemize}
  \item \texttt{dsort} \textit{st label info inf} stands for \texttt{sorts} \textit{st, lab, inf}
  \item \texttt{pred \_ : \_ } \rightarrow \texttt{\_ : inf \times st \times lab \times st}
\end{itemize}

- \texttt{poSpec.parts} = \{ \textit{ds}_1, \ldots, \textit{ds}_j \}, and that \textit{DS}_1, \ldots, \textit{DS}_j are the CASL-LTL presentations of the data structure specifications \textit{ds}_1, \ldots, \textit{ds}_j respectively.

- \texttt{poSpec.subsyst-Specs} = \{ \textit{ssp}_1, \ldots, \textit{ssp}_k \}, that \textit{SSP}_1, \ldots, \textit{SSP}_k are the CASL-LTL presentations of the system specifications \textit{ssp}_1, \ldots, \textit{ssp}_k respectively, and that \texttt{ELINTERACTION}_1, \ldots, \texttt{ELINTERACTION}_k be the specifications of their elementary interactions.

- \texttt{poSpec.e-features} = \{ \textit{ei}_1, \ldots, \textit{ei}_n \} the elementary interactions

- \texttt{poSpec.s-features} = \{ \textit{so}_1, \ldots, \textit{so}_m \} the state observers

- \texttt{poSpec.subsystems} = \{ \textit{ss}_1, \ldots, \textit{ss}_r \} the subsystems
\textbf{CASL-LTL View foll’d (Structured/Property)}

\textbf{spec} \textbf{LOCAL}\textbf{INTERACTION} =

\textbf{EL}\textbf{INTERACTION}_1 \textbf{ and } \ldots \textbf{ and } \textbf{EL}\textbf{INTERACTION}_k \textbf{ and } \textbf{Ident} \textbf{ then}

\textbf{free type} \textit{subEllInteraction} ::= \textopen{(ellInteraction}_1 \textclose{)} | \ldots | \textopen{(ellInteraction}_k \textclose{)}

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

\textbf{free type} \textit{localInteraction} ::= \textopen{< } \textopen{, } \textopen{ > } (\textit{ident, subEllInteraction})

\textbf{spec} \textit{poSpec}\textbf{.name} =

\textbf{FINITE}\textbf{SET}[\textit{ELL}\textbf{INTERACTION}] \textbf{ and } \textbf{FINITE}\textbf{SET}[\textit{LOCAL}\textbf{INTERACTION}] \textbf{ and }

\textbf{DS}_1 \textbf{ and } \ldots \textbf{ and } \textbf{DS}_j \textbf{ and } \textbf{SSP}_1 \textbf{ and } \ldots \textbf{ and } \textbf{SSP}_k \textbf{ then}

\textbf{dsort} \textit{st label} \textbf{FinSet[ellInteraction]} \textbf{ info} \textbf{FinSet[localInteraction]}

\textbf{ops} \textit{so_a}.name : \textit{st} \times \textit{so_a}.\textbf{argTypes} \rightarrow \textit{so_a}.\textbf{resType} \textbf{ %\% state observers}

\ldots

\textit{so_m}.name : \textit{st} \times \textit{so_m}.\textbf{argTypes} \rightarrow \textit{so_m}.\textbf{resType}

\textit{ss_1}.id : \textit{st} \rightarrow \textit{ss_1}.\textbf{type} \textbf{ %\% observers of the subsystem states}

\ldots

\textit{ss_r}.id : \textit{st} \rightarrow \textit{ss_r}.\textbf{type}

\textbf{axioms} \textit{those formulae corresponding to the cell fillings}

\textbf{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
Data Structure Items / Property

- Constituent Feature
- Data structure
  - features
  - parts
- Constructor
- Operation
- Predicate

Property-oriented

- Constructor definition name: String
  argTypes: Sequence(Type)
  c-features
  - Property
    properties
  - properties
- Predicate definition name: String
  argTypes: Sequence(Type)
  p-features
- Operation definition name: String
  argTypes: Sequence(Type)
  resultType: Type
  o-features
- Data structure specification parts

Data structure property-oriented specification
name: String

Towards a Formally Grounded Development Method
Data Structure - Property-oriented

DataStructureName

- Constructors
  - con(type1, ..., typen) or con(type1, ..., typen)?

- Predicates
  - pr(type1, ..., typen)

- Operations
  - op(type1, ..., typen): type or op(type1, ..., typen): ? type

Towards a Formally Grounded Development Method
Floor (Data Structure/Property-oriented)

<table>
<thead>
<tr>
<th>Floor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ground</td>
</tr>
<tr>
<td>top</td>
</tr>
<tr>
<td>_ above _(Floor,Floor)</td>
</tr>
<tr>
<td>next(Floor): ? Floor</td>
</tr>
<tr>
<td>previous(Floor): ? Floor</td>
</tr>
</tbody>
</table>

- 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 \( \text{next}(f) \).

  ```python
  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 \( f_1 \) \( (\text{next}(f) \text{ above } f_1 \text{ and } f_1 \text{ above } f) \)
  whenever everything is defined \( \text{next}(\text{previous}(f)) = \text{previous}(\text{next}(f)) = f \)
  
- `above` is total order over the floors with `top` as maximum and `ground` as minimum ....
**CASL View (Data/Property)**

- \( poSpec.\text{parts} = \{ds_1, \ldots, ds_j\} \) w/ \( DS_1, \ldots, DS_j \) CASL-LTL presentations
- \( poSpec.\text{c-features} = \{con_1, \ldots, con_n\} \) the constructors
- \( poSpec.\text{o-features} = \{op_1, \ldots, op_m\} \) the operations
- \( poSpec.\text{p-features} = \{pr_1, \ldots, pr_p\} \) the predicates.

\[
\text{spec } poSpec.\text{name} = \\
\quad DS_1 \quad \text{and} \ldots \quad \text{and} \quad DS_j \quad \text{then} \\
\quad \text{type } poSpec.\text{name} ::= \ con_1.\text{name}(con_1.\text{argTypes})? | \ldots | con_n.\text{name}(con_n.\text{argTypes}) \\
\quad \text{ops } op_1.\text{name} : op_1.\text{argTypes} \rightarrow? op_1.\text{resType} \\
\quad \ldots \\
\quad \quad op_m.\text{name} : op_m.\text{argTypes} \rightarrow op_m.\text{resType} \\
\quad \text{preds } pr_1.\text{name} : pr_1.\text{argTypes} \\
\quad \ldots \\
\quad \quad pr_p.\text{name} : pr_p.\text{argTypes} \\
\quad \text{axioms}
\]

\[\text{formulae corresponding to the cell fillings}\]
Data Structure - Constructive

- Constructor definition
  - c-features
  - ConditionalRule
    - conditional-rules
  - o-features
    - Operation definition
      - p-features
    - Data structure specification
      - parts

Data structure constructive specification

name: String

Towards a Formally Grounded Development Method

Jan 2005
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")

Towards a Formally Grounded Development Method
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)

Towards a Formally Grounded Development Method
... 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, ...)

Towards a Formally Grounded Development Method
Part II

Integrate the specification development method together with use cases requirement description

- Some description is required before a specification may be written
- Is it possible to establish a connection between both?
- Guidelines for these tasks
HOW? (1)

- **Use USE CASES**
  - **use case =**
    - description of interactions between the system under discussion and external actors, related to the goal of one particular actor
    - description is textual ("familiar", easy to read) and sums up a set of scenarios (sequences of interactions between system and actors)
  - quite successful
    - easy to use and informal
    - easily give an idea of the system that can be discussed with the client
    - a lot of freedom in what should include a use case description, and how it should be written
  - however
    - "use cases are wonderful but confusing" (Cockburn 2000)
    - use cases are often imprecise, and used terms are vague or ambiguous
HOW ? (2)

• Use formal specifications
  – lead to precise, unambiguous descriptions
  – but difficult to use and impractical in quite a number of cases
  – hard to write/read these specifications
  – hard to start with formal specifications while still working on the requirements (thus, trying to understand what is the problem about)
HOW ? (3)

• combine advantages of use cases and of formal specifications
  – improving use case based requirements by developing a companion Formally Grounded specification [ChoppyReggio2003]
    • written in a “visual” notation (diagrams and text)
    • with a formal counterpart written in the logical-algebraic CASL-LTL specification language
    • produced following a systematic method, arising questions on all the aspects of the specified system
  – resulting in
    • requirement validation, writing the Formally Grounded specification leads to thoroughly check that requirements
    • improved requirements (requirements may be updated)
    • improved use case based requirement specification
    • a formal specification available for formal analysis
Case study: Auction System

- Online auction system to allow to buy/sell goods
- Innovative because it guarantees that bid placed are solvent
- Users must first enroll and log on for each session, then they are able to sell, buy, or browse the available auctions
- Customers have credit with the system used as security on each bid; and can increase it by asking the system to debit a certain amount from their credit card, and when sell
- A customer that wishes to sell initiates an auction by informing the system of the goods to auction with ....
- Customers that wish to follow an auction must first join the auction, then they may make a bid, or post a message
- Bidders are allowed to place their bids until the auction closes, and place bids across as many auctions as they please
Auction System: task 1

Give a Use case based requirement specification

- (UML) Use case diagram

- Use case descriptions
  (S. Sendall and A. Strohmeier template)
Use Case buy item under auction

Intention in Context: The intention of the Customer is to follow the auction, ...

Primary Actor: Customer

Precondition: Customer already identified

Main Success Scenario:
1. Customer searches for an item under auction (search item).
2. Customer requests to join the item auction.
3. System presents a view of the auction
...

buy item under auction (contd)

Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer

4. Customer makes a bid on the item to System.

5. System validates the bid, records it, secures the bid amount from Customer's credit, ... informs Participants of new high bid, and updates the view of the auction

6. System closes the auction with a winning bid by Customer.

Extensions: ...
buy item under auction (extens.)

Extensions:

2a. C requests System not to pursue item further:
   2a.1. System permits Customer to choose another auction, or go back to an earlier point in the selection process; uc continues at step 2.

3a. System informs Customer that auction has not started: use case ends in failure.

3b. System informs Customer that auction is closed: use case ends in failure.

...
Auction System: task 2

By looking at the Use case diagram give

- **Context View (initial version)**

![Diagram](image)
Auction System: task 3

By looking at use case descriptions one after the other (here Buy Item under Auction) give

- AuctionSystem specification interface

- simple dynamic system characterized by its states and labelled transitions
- labelled transition = state change + label (set of elementary interactions with external world)
- states abstractly characterized by “state observers”
Auction System: task 3 (cont.)

- Data View

Data used to type parameters and results of state observers and elementary interactions.
Auction System: task 4

- find the **properties** about AuctionSystem by filling “forms” generated by the elementary interactions and state observers found in the previous task **systematically covering** “all” its aspects

  - based on a many-sorted, first-order, CTL*-style temporal logic with edge formulae

- In the meantime
  - previous diagrams may be modified
  - new state observers may be added
  - original use case based requirement specification may be modified to reflect the better insights on the AuctionSystem gained while looking for properties
(sample) Properties on CUSTOMER_JOIN_AUCTION

Form fragment

• pre/postcondition

if CUSTOMER_JOIN_AUCTION(sk) happen then

…condition about state observers on source state (of any transition having that elementary interaction in its label) …

if CUSTOMER_JOIN_AUCTION(sk) happen then

…condition about state observers target states (of any transition having that elementary interaction in its label) …

• Problems/Questions

– Does the included use case search item ends having selected one auction or one item?
– Can an auction selected by search item be in any status (e.g., closed or not yet started)?
– Can a Customer try to join a closed or not-started auction?
– Can a Customer join an auction to which (s)he is already joined?
(sample) Properties on CUSTOMER_JOIN_AUCTION

if CUSTOMER_JOIN_AUCTION(sk) happen then

exists id:Identification s.t. is_Identified(id,sk) and
exists aid:Auction_Id s.t.

selected_Auctions(sk) = \{aid\} and
status(infoAbout(aid)) = active and
joined^{nxt}(sk,aid) and

in any case next

AS_SHOW_AUCTION(sk,view(infoAbout(aid))) happen
(sample) Properties on credit

Form fragment

- how change

\[
\text{if } \text{credit}(id) = x \text{ and } \text{credit}^{\text{nxt}}(id) = y \text{ and } x \neq y \text{ then}
\]

...condition about id, x and y and some elementary interactions must happen in that transition (belong to its label) ...

Property

\[
\text{if } \text{credit}^{\text{nxt}}(id) = \text{credit}(i) - i \text{ and } i > 0 \text{ then}
\]

exists sk:SessionKey, ai:AuctionId s.t.

\text{AS\_BID\_OK}(sk,ai,i) \textbf{happen} \textbf{and} \text{is\_Identified}(id,sk)

- Problems/Questions

  - It is true that a Customer using the AuctionSystem only for selling items will be never able to collect her/his money? Moreover, can a buying Customer recover her/his money when (s)he is no more interested in buying?
Auction System: task 5

Revised Use case based requirement specification

New Use case diagram
Revised “buy item under auction”

Intention in Context: The intention of the Customer is to follow the auction, ...

Primary Actor: Customer

Precondition: Customer already identified and selected one active auction NEW

Main Success Scenario:

1. Customer searches for an item under auction (search item). REMOVED

2. Customer requests to join the item auction.

3. System presents a view of the auction

...
buy item under auction (contd)

Steps 4-5 can be repeated according to the intentions and bidding policy of the Customer.

4. Customer makes a bid on the item to System.

5. System validates the bid, records it, secures the bid amount from Customer's credit, … informs Participants of new high bid \textbf{REMOVED}, and updates the view of the auction.

6. System closes the auction with a winning bid by Customer.

\textbf{Extensions: ...}
buy item under auction (extens.)

Extensions:

2a. C requests System not to pursue item further:
   2a.1. System permits Customer to choose another auction, or go
        back to an earlier point in the selection process; uc continues
        at step 2.

3a The Customer is the Seller of the auction; System
   informs Customer that (s)he cannot join the auction.
   Use case ends with failure NEW

3a. System informs Customer that auction has not started:
    use case ends in failure. REMOVED

3b. System informs Customer that auction is closed: use
    case ends in failure. REMOVED

...
Conclusion

• proposed a method to review use case based requirements by building a companion Formally Grounded specification
  – as result
    • initial requirements examined in a systematic way by looking at the various aspects of the considered system
    • original use case based requirements updated whenever an aspect of the system is enlightened
    • the Formally Grounded specification (diagrams plus textual annotations) could be used as an alternative requirement document
    • the CASL-LTL specification corresponding to the Formally Grounded one is also available, e.g., for formal analysis
• building directly the Formally Grounded specification not as much as effective as the proposed combination
  – Formally Grounded specification ingredients (elementary interactions and state observers) finer grained than system functionalities, thus hard to find them just considering the problem
Auction System Experiment

• medium-size case study
• starting use case requirements
  – not produced by ourselves
  – quite accurate and presented using a well-organized template
• positive outcome
  – detected many problematic or unclear aspects in the original use case based Requirements
    • explicit auctions browsing functionality
    • auctions should be performed in a chat-like way
    • need for a decrease-credit functionality
    • two different Customers may be the same person
    • a Customer may disconnect by the System by hers\his own choice, and not only after sometime (s)he is doing nothing
    • a Customer cannot unregister from the System when (s)he is the seller or has the high bid in an auction
    • made explicit that when a Customer unregisters any left credit is seized by the Auction System owner
• …
Related work: inspection techniques

• Inspection techniques for requirement spec: ad hoc techniques or check-lists
  “Is there any missing functionality, that is, do the actors have goals that must be fulfilled, but that have not been described in use cases?”

• Our “inspection”: build a companion formal specification with a form-filling technique leads to a systematic and precise requirement examination
  “find and list all the ways the credit state observer may be updated in the various scenarios of all use case”
  -> credit decreasing needed !!
Our high quality requirements method

Task 1: use case diagram & descriptions (Sendall & Strohmeier)

Iterative construction of the specification:

Task 2: initial Context View
configuration diagram & cooperation diagram

Task 3: for each use case
- elementary interactions & state observers
  -> cooperation diagram (update)
- Data View (data structures)

Task 4: properties (form filling method)
  -> update elem inter, state obs, data struct

Task 5: in parallel, record questions
  -> update use case accordingly
and more …

- General requirement formal specification development method
- Initially aimed for CASL/CASL-LTL languages
- Could be used with other specification languages (colored/high level Petri nets, …)
- May be used in combination with informal notations/methods: use cases, UML, problem frames, …
- Architectural styles may be used to work further towards the design specification
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 reference 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
Different aims

Related work

• formal specification of requirements, e.g.
  – A. van Lamsweerde and his group
    • formally specifications of goal-oriented requirements plus
      analysis by means of formal techniques
  – R. Drome
    • “Behaviour Tree” a formal-visual notation to specify the
      requirements, and a method to derive from them the
      architectural structuring of the system

• “more precise” specification of requirements, e.g.
  – S. Sendall and A. Strohmeier
    • operation schemas (written in OCL) and system interface
      protocols (UML statecharts) to complement use cases
  – E. Astesiano- G. Reggio
    • Tight-structured UML based method for the precise
      specification of the requirements, where use case are
      modelled by statecharts