Synthesizing B specifications from EB³ attribute definitions

Frédéric Gervais¹,², Marc Frappier¹, Régine Laleau³

¹ GRIL, Département d'Informatique, Université de Sherbrooke
² CEDRIC, Institut d'Informatique d'Entreprise
³ LACL, Université Paris 12
Specification of IS

• Main issues:
  – Consistency of the DB: data integrity constraints
  – Answer of the IS
  – Modeling of the IS behaviour

• Aim: use of formal notations and techniques

• Coupling of formal specifications?
  – B: state-based formal language
  – EB^3: event-based formal language
• State-based formal language

• Based on set theory and first order logic:
  – Static part: definition of state variables and invariant properties
  – Dynamic part: definition of operations (preconditions + substitutions)
  – Proof obligations: the invariant must be preserved by the operations

• Refinement

• Tools: Atelier B, B4Free, Click'n'Prove, B-Core
EB$^3$

- Trace-based formal language
- Inspired from process algebras (CSP, CCS, LOTOS)
- Specifically created for the design of information systems
- Specification of the functional behaviour of the system
- Interpreter (EB3PAI)
EB$^3$ specification

- User requirements class diagram (entity types, associations)
- EB$^3$ process expressions
- Input-output rules
- Recursive functions defined on EB$^3$ traces (attribute definitions, auxiliary functions)
Behaviour

\begin{verbatim}
trace := []; 
forever do 
    receive input event \( \sigma \);
    if main can accept trace::\( \sigma \) then 
        trace := trace::\( \sigma \);
        send output event o such that (trace,o) \( \in \) \( R \);
    else
        send error message ;
\end{verbatim}
Class diagram

**book**
- bookKey : bk_Set
- title : T
- Acquire
- Discard
- Modify
- DisplayTitle

**member**
- memberKey : mk_Set
- nbLoans : NAT
- Register
- Unregister

**loan**
- Lend
- Return
- Transfer

*Acquire(bId : bk_Set, ttl : T) : void
Modify(bId : bk_Set, ttl : T) : void
...
EB\textsuperscript{3} process expressions

\begin{verbatim}
book(bId : bk_Set) =
    Acquire(bId, _).
    ( |
        mId : mk_Set : loan(bId, mId) )*
    |||
        Modify(bId, _)*
    |||
        DisplayTitle(bId)*
    ).
Discard(bId)
\end{verbatim}
EB³ process expressions

book(bId : bk_Set) =
    Acquire(bId, _).
    ( ( | mId : mk_Set : loan(bId, mId) )* 
        ||
        Modify(bId, _)* 
        ||
        DisplayTitle(bId)* 
    ).
Discard(bId)

RULE R1
    INPUT DisplayTitle(bId)
    OUTPUT title(trace, bId)
END;
EB³ attribute definitions

- Recursive functions on the valid traces
- Two kinds of definitions:
  - Key definition: set of the existing key values for the key
  - Non-key attribute definition: the attribute value for a given key
- Specification: functional style with pattern matching on the last event of the trace
Examples

bookKey(s : τ(main)) : \|F(bk_Set) =
match last(s) with
  \bot : \emptyset,
  Acquire(bId,ttl) : bookKey(front(s)) \cup \{ bId \},
  Discard(bId) : bookKey(front(s)) - \{ bId \},
  _ : bookKey(front(s));

title(s : τ(main), bId : bk_Set) : T =
match last(s) with
  \bot : \bot,
  Acquire(bId,ttl) : ttl,
  Modify(bId,ttl) : ttl,
  Discard(bId) : \bot,
  _ : title(front(s),bId);
Aim: define a formal approach coupling state-based and event-based specifications for IS

• $EB^4$:
  - Trace-based formal language for IS specification
  - Description of the functional behaviour

• $EB^3$:
  - State-based formal language
  - Description of static data integrity constraints
  - Theorem prover
EB³ Abstract Specification

EB³ translatable specification

Process Expressions

Attribute Definitions

EB³ Refinement

Translation into B

B Model of Process Expressions

B Specification

Specification of the Data Model

B Refinement

Implementation

EB⁴
EB^3 Abstract Specification

EB^3 Refinement

EB^3 translatable specification

Process Expressions

Attribute Definitions

Translation into B

B Model of Process Expressions

B Specification

Specification of the Data Model

B Refinement

Implementation

B Refinement

EB^4
B specification in 4 steps

- Generation of the static part from the user requirements class diagram
- Generation of the substitutions for the initialization
- Generation of the substitutions for the operations
- Computation of the preconditions for the operations
B static part

MACHINE  B_Library
SETS  mk_Set; bk_Set; T
VARIABLES  memberKey, bookKey, loan, title, nbLoans
INVARIANT
  memberKey \in mk_Set \land
  bookKey \in bk_Set \land
  loan \in bookKey \rightarrow memberKey \land
  title \in bookKey \rightarrow T \land
  nbLoans \in memberKey \rightarrow \text{NAT}
DEFINITIONS
  borrower(x) == loan(x)
B substitutions

• Orthogonal specification styles:
  – $\text{EB}^3$ attribute definitions: for each attribute, effects of the actions
  – B specifications: for each operation, substitutions on the attribute variables

• Analysis of the input clauses:
  – Determine each occurrence of the action
  – Determine the relevant key values (pattern matching + predicates in conditional terms)
  – Determine the effects on the attributes
Example: Acquire

bookKey(s : τ(main)) : \mathcal{F}(bk_\text{Set}) =
\textbf{match} \text{ last}(s) \text{ with}
\begin{align*}
\bot & : \emptyset, \\
\text{Acquire}(bId,ttl) & : \text{bookKey}(\text{front}(s)) \cup \{ bId \}, \\
\text{Discard}(bId) & : \text{bookKey}(\text{front}(s)) - \{ bId \}, \\
_ & : \text{bookKey}(\text{front}(s));
\end{align*}

title(s : τ(main), bId : bk_\text{Set}) : T =
\textbf{match} \text{ last}(s) \text{ with}
\begin{align*}
\bot & : \bot, \\
\text{Acquire}(bId,ttl) & : ttl, \\
\text{Modify}(bId,ttl) & : ttl, \\
\text{Discard}(bId) & : \bot, \\
_ & : \text{title}(\text{front}(s),bId);
\end{align*}
Example: Acquire

\[ \text{Acquire}(bld, ttl) = \]
\[ \text{pre } \quad bld \in bk\_Set \land ttl \in T \]
\[ \text{then} \]
\[ \quad \text{bookKey} \ := \ \text{bookKey} \cup \{ bld \} || \]
\[ \quad \text{title} \ := \ \text{title} \leftarrow \{ bld \mapsto ttl \} \]
\[ \text{end}; \]
Example: Discard

\[ \text{bookKey}(s : \tau(\text{main})) : \mathcal{F}(\text{bk}_\text{Set}) = \]

**match** last(s) **with**

- \( \bot \) : \( \emptyset \),
- Acquire(bId,ttl) : bookKey(front(s)) \( \cup \) \{ bId \},
- Discard(bId) : bookKey(front(s)) \( - \) \{ bId \},
- \_ : bookKey(front(s));

\[ \text{title}(s : \tau(\text{main}), \text{bId} : \text{bk}_\text{Set}) : T = \]

**match** last(s) **with**

- \( \bot \) : \( \bot \),
- Acquire(bId,ttl) : ttl,
- Modify(bId,ttl) : ttl,
- Discard(bId) : \( \bot \),
- \_ : title(front(s),bId);
Example: Discard

\[\text{Discard}(\text{bld}) =\]
\[
\text{pre } \text{bld} \in \text{bk}\_\text{Set} \]
\[
\text{then}\]
\[
\quad \text{bookKey} := \text{bookKey} - \{ \text{bld} \} \parallel \]
\[
\quad \text{title} := \{ \text{bld} \} \triangleq \text{title} \]
\[
\text{end};\]
Conditional terms

- The key values to delete, to update or to insert are determined by the pattern matching condition and the predicates in the if parts
- Generation and analysis of a decision tree
Transfer

\[ \text{nbLoans}(s : \tau(\text{main}), \text{mId} : \text{mk}\_\text{Set}) : \text{NAT} = \]

\[ \text{match last}(s) \text{ with} \]

\[ \bot : \bot, \]

\[ \text{Register(\text{mId})} : 0, \]

\[ \text{Unregister(\text{mId})} : \bot, \]

\[ \text{Lend(_,mId)} : 1 + \text{nbLoans(front}(s),\text{mId}), \]

\[ \text{Return(bId)} : \text{if mId} = \text{borrower(front}(s),\text{bId}) \]

\[ \text{then nbLoans(front}(s),\text{mId}) - 1 \]

\[ \text{end}, \]

\[ \text{Transfer(bId,mId')} : \text{if mId} = \text{mId}' \]

\[ \text{then nbLoans(front}(s),\text{mId}) + 1 \]

\[ \text{else if mId} = \text{borrower(front}(s),\text{bId}) \]

\[ \text{then nbLoans(front}(s),\text{mId}) - 1 \]

\[ \text{end} \]

\[ \text{end}, \]

\[ \bot : \text{nbLoans(front}(s),\text{mId}); \]
Transfer

\[
\text{Transfer}(\text{bId, mId}) : \text{if mId} = \text{mId'} \\
\quad \text{then} \ \text{nbLoans(front(s),mId)} + 1 \\
\quad \text{else if mId} = \text{borrower(front(s),bId)} \\
\quad \quad \text{then} \ \text{nbLoans(front(s),mId)} - 1 \\
\quad \quad \text{end} \\
\quad \text{end} \\
\quad : \text{nbLoans(front(s),mId)};
\]

\[
\text{nbLoans}(s : \tau(\text{main}), \text{mId : mk_Set}) : \text{NAT} = \\
\text{match last(s) with} \\
\quad \bot : \bot, \\
\quad \text{Register(mId)} : 0, \\
\quad \text{Unregister(mId)} : \bot, \\
\quad \text{Lend(\_ , mId)} : 1 + \text{nbLoans(front(s),mId)}, \\
\quad \text{Return(bId)} : \text{if mId} = \text{borrower(front(s),bId)} \\
\quad \quad \text{then} \ \text{nbLoans(front(s),mId)} - 1 \\
\quad \quad \text{end} \\
\quad \text{Transfer(bId,mId') : if mId} = \text{mId'} \\
\quad \quad \text{then} \ \text{nbLoans(front(s),mId)} + 1 \\
\quad \quad \text{else if mId} = \text{borrower(front(s),bId)} \\
\quad \quad \quad \text{then} \ \text{nbLoans(front(s),mId)} - 1 \\
\quad \quad \quad \text{end} \\
\quad \quad \text{end} \\
\quad : \text{nbLoans(front(s),mId)};
\]
Decision tree

- \( nbLoans \)
  - \( mId = mId' \)
    - \( nbLoans(mId) + 1 \)
  - \( mId \neq mId' \)
    - \( mId = \text{borrower}(bId) \)
      - \( nbLoans(mId) - 1 \)
    - \( mId \neq \text{borrower}(bId) \)
      - Recursive call
Precondition

- Typing constraints for input parameters
- The weakest precondition required to preserve the invariant of the machine
- Condition required to impose ordering constraints on operations
Invariant

\[ \text{Discard}(bld) = \]
\[
\text{pre } bld \in bk\_Set \\
\text{then} \\
\quad \text{bookKey} := \text{bookKey} - \{bld\} || \\
\quad \text{title} := \{bld\} \triangleq \text{title} \\
\text{end;} \\
\]

loan \in \text{bookKey} \leftrightarrow \text{memberKey}
Invariant

\[ \text{Discard}(bld) = \]
\[ \text{pre} \quad bld \in bk\_Set \land bld \notin \text{dom}(loan) \]
\[ \text{then} \]
\[ \quad \text{bookKey} := \text{bookKey} - \{ bld \} \parallel \]
\[ \quad \text{title} := \{ bld \} \cup \text{title} \]
\[ \text{end;} \]

\[ \text{loan} \in \text{bookKey} \leftrightarrow \text{memberKey} \]
Weakest precondition

• Issue: determining the weakest precondition from the invariant and the operation substitutions
• Solution: using the logic solver of Atelier B
• Weakness: interactive use of the prover
• Work in progress: definition of systematic rules
EB³ Abstract Specification

EB³ Refinement

EB³ translatable specification

EB³ Refinement

Process Expressions

Translation into B

Attribute Definitions

Specification of the Data Model

EB⁴ Refinement

B Model of Process Expressions

B Specification

B Refinement

Implementation
It's almost finished!

Discard(bId) =
pre bId ∈ bk_Set ∧ bId ∉ dom(loan) ∧ bId ∈ bookKey
then
  bookKey := bookKey - { bId } ||
  title := { bId } ⊑ title
end;

book(bId : bk_Set) =
  (Acquire(bId, _).
   ( | mId : mk_Set : loan(bId, mId) )*.
    Discard(bId)
  )*
Conclusion

- Generation of B substitutions from EB$^3$ attribute definitions
- Generation of the operation preconditions:
  - Weakest preconditions
  - Proof of static data integrity constraints
- EB$^4$: IS specification by coupling EB$^3$ and B
Perspectives

• Definition of systematic rules for the weakest preconditions
• Definition of refinement patterns
• $\text{EB}^4$:
  - Definition of a methodology
  - Definition of verification techniques
  - Implementation of tools
  - Integration between $\text{EB}^3$ and B
Questions?