A Functional Language for Specifying Business Reports

Patrick Bahr

University of Copenhagen, Department of Computer Science
paba@diku.dk

23rd Nordic Workshop on Programming Theory,
Mälardalen University, Västerås, Sweden,
October 26 - 28, 2011
Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions
Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions
What are Enterprise Resource Planning Systems?

ERP systems integrate several software components that are essential for managing a business.
What are Enterprise Resource Planning Systems?

ERP systems integrate several software components that are essential for managing a business.

- Financial Management
- Supply Chain Management
- Manufacturing Resource Planning
- Human Resource Management
- Customer Relationship Management
- ...
What are Enterprise Resource Planning Systems?

ERP systems integrate several software components that are essential for managing a business.

ERP systems integrate:
- Financial Management
- Supply Chain Management
- Manufacturing Resource Planning
- Human Resource Management
- Customer Relationship Management
- ...
What do ERP Systems Look Like?
# Issues of Many ERP Implementations

## Complexity

- Processes are specified in **general purpose language**
- Gap between specification and implementation
- Large **monolithic system**

---
Issues of Many ERP Implementations

**Complexity**
- processes are specified in **general purpose language**
- gap between specification and implementation
- large **monolithic** system

**Inflexibility**
- code is duplicated in order to avoid unexpected side effects
- the use of general purpose languages makes **customisation expensive**
- the (relational) database determines the way data is stored and accessed
Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions
Entering POETS
Process-oriented event-driven transaction systems

compact core system
Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs
Entering POETS
Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model
Entering POETS
Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model
Entering POETS

Process-oriented event-driven transaction systems

- compact core system • customisable via DSLs • simple data model
Entering POETS
Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model
Entering POETS
Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model

Contract engine
- Running contracts
  - start contract
  - register event
  - end contract

Event log
- events
- updates

Report engine
- Report definitions
  - add/delete report
  - modify report
  - query report

query results
Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model

Contract engine
- Running contracts
  - start contract
  - register event
  - end contract

Event log
- events
- updates
- query results

Report engine
- Report definitions
  - add/delete report
  - modify report
  - query report
What is a Report?

Event Log

- event 1
- event 2
- event 3
- event 4
- event 5
- event 6
- event 7
- event 8
- event 9
- event 10
What is a Report?

Event Log

- event 1
- event 2
- event 3
- event 4
- event 5
- event 6
- event 7
- event 8
- event 9
- event 10

Report Function

```plaintext
invoices : [Invoice]
invoices = [ Invoice{
    customer = ii.customer@,
    orderLines = ii.orderLines} |
    tr : TransactionEvent ← events,
    ii : IssueInvoice = tr.transaction]
```
What is a Report?

**Event Log**

- event 1
- event 2
- event 3
- event 4
- event 5
- event 6
- event 7
- event 8
- event 9
- event 10

**Report Function**

```lua
invoices : [Invoice]
invoices = [ Invoice{
  customer = ii.customer@,
  orderLines = ii.orderLines} |
  tr : TransactionEvent ← events,
  ii : IssueInvoice = tr.transaction]
```

**Report**

```plaintext
invoice 1 {
  customer = {...},
  orderLines = {...}
}

invoice 2 {
  customer = {...},
  orderLines = {...}
}

invoice 3 {
  customer = {...},
  orderLines = {...}
}
```
What is a Report?

**Event Log**

- event 1
- event 2
- event 3
- event 4
- event 5
- event 6
- event 7
- event 8
- event 9
- event 10

**Report Function**

```plaintext
invoices : [Invoice]
invoices = [ Invoice{
    customer = ii.customer@,
    orderLines = ii.orderLines} |
    tr : TransactionEvent ← events,
    ii : IssueInvoice = tr.transaction]
```

**Report**

- invoice 1 {
  customer = {...},
  orderLines = {...}
}
- invoice 2 {
  customer = {...},
  orderLines = {...}
}
- invoice 3 {
  customer = {...},
  orderLines = {...}
}
The Report Language

The central data types

- records
- lists
The Report Language

The central data types

- records: events are records
- lists
The Report Language

The central data types

- **records**: events are records
- **lists**: the event log is a list of events
The Report Language

The central data types

- **records**: events are records
- **lists**: the event log is a list of events

Nominal subtyping

![Event subtyping diagram]

- Event
  - Entity Event
    - Put Entity Event
    - Delete Entity Event
  - Contract Event
  - Report Event
The Report Language

The central data types

- records: events are records
- lists: the event log is a list of events

Nominal subtyping

```
Event
  Entity Event
    Put Entity
    Delete Entity
  Contract Event
  Report Event

Transaction
  Transfer
    Payment
    Delivery
  Issue Invoice
```
The Report Language – An Example Function

Example

reportNames : [String]
reportNames = [pr.name | 
cr : CreateReport ← events,
pr : PutReport = head [ur | 
   ur : ReportEvent ← events,
   ur.id ≡ cr.id]
]
The Report Language – An Example Function

**Example**

```plaintext
reportNames : [String]
reportNames = [pr.name |
    cr : CreateReport ← events,
    pr : PutReport = head [ur |
        ur : ReportEvent ← events,
        ur.id ≡ cr.id]
]```

**Report Event Hierarchy**

- **Report Event**
  - **Put Report**
  - **Delete Report**
  - **Create Report**
  - **Update Report**
Nominal Subtyping with Benefits

<table>
<thead>
<tr>
<th>Nominal subtype relation $\ll$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>User defined subtyping partial order on records</td>
</tr>
<tr>
<td>Fixed subtyping relation on built-in types</td>
</tr>
</tbody>
</table>
Nominal Subtyping with Benefits

Nominal subtype relation <:

- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

Record Constraints

\[ \tau_1.f : \tau_2 \]
Nominal Subtyping with Benefits

Nominal subtype relation $\prec$:
- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

Record Constraints

$\tau_1 . f : \tau_2$

E.g. field selector operator $_.f$ has type

$\alpha . f : \beta \Rightarrow \alpha \rightarrow \beta$
# Nominal Subtyping with Benefits

## Nominal subtype relation $\prec$:
- User defined subtyping partial order on records
- Fixed subtyping relation on built-in types

## Record Constraints

$$\tau_1.f : \tau_2$$

E.g. field selector operator $_.f$ has type

$$\alpha.f : \beta \Rightarrow \alpha \rightarrow \beta$$

E.g. record modifier operator $\_ \{ f_1 = _, \ldots, f_n = _ \}$ has type

$$\alpha.f_1 : \alpha_1, \ldots, \alpha.f_n : \alpha_n \Rightarrow \alpha \rightarrow \alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \alpha$$
Record Field Constraints

What do we gain?

- Field names can be used by different record types.
- Nominal subtyping feels like structural subtyping (unless you want to create a record).
Record Field Constraints

What do we gain?

- Field names can be used by different record types.
- Nominal subtyping feels like structural subtyping (unless you want to create a record).

Example

```haskell
fullName : (a.firstName : String, a.lastName : String) ⇒ a → String
fullName x = x.firstName ++ " " ++ x.lastName
```
Record Field Constraints

What do we gain?

- Field names can be used by different record types.
- Nominal subtyping feels like structural subtyping (unless you want to create a record).

Example

```plaintext
fullName : (a.firstName : String, a.lastName : String) ⇒ a → String
fullName x = x.firstName ++ " " ++ x.lastName

setFullName : (a.firstName : String, a.lastName : String) ⇒ String → a → a
setFullName name x = let (first,last) = decompose name
                   in x {firstName = first, lastName = last}
```
Making It Scale

ain’t easy

\[
\begin{array}{|c|}
\hline
\text{event 1} \\
\hline
\text{event 2} \\
\hline
\text{event n} \\
\hline
\end{array}
\]
Making It Scale

ain’t easy

\[
\text{event 1} \quad \text{event 2} \quad \text{event n}
\]

\[
\text{report function } r
\]

\[
\text{Report } R
\]

\[
\text{old result } + \Delta
\]

\[
\text{incrementalised variant of } r
\]
Making It Scale

ain’t easy

event 1
event 2
event n
event n+1

\begin{align*}
\text{report function } r
\end{align*}

\begin{align*}
\text{Report } R
\end{align*}
Making It Scale

ain't easy

\[ \text{event 1} \]
\[ \text{event 2} \]
\[ \text{event } n \]
\[ \text{event } n+1 \]

report function \( r \)

Report \( R \)

obsolete
Making It Scale

ain’t easy

event 1
event 2

\{ 
\}

event n

\underline{event n+1}

report function \( r \)

Report \( R \)

\[ \text{obsolete} \]

\[ \text{Report } R' \]

\[ \text{too expensive!} \]
Making It Scale

ain’t easy

\[
\text{event 1} \\
\text{event 2} \\
\text{event } n \\
\text{event } n+1
\]

\[
\text{report function } r
\]

\[
\text{too expensive!}
\]

\[
\text{Report } R \\
\text{obsolete}
\]

\[
\text{Report } R'
\]

\[
\text{old result } + \Delta
\]
Making It Scale

ain’t easy

\[
\text{event 1} \\
\text{event 2} \\
\text{event } n \\
\text{event } n+1
\]

\[
\text{report function } r \\
\text{old result } + \Delta \\
\text{Report } R \\
\text{Report } R'
\]

"obsolete"
Making It Scale

ain't easy

event 1
event 2

\{ event n \\
\textbf{event n+1} \}

\text{report function } r

\text{old result } + \Delta

\text{incrementalised variant of } r

\text{Report } R

\text{obsolete}

\text{Report } R'
Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

\[
\text{fold } f \ e \ (x \# \ xs)
\]
Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

\[
\text{fold } f \ e \ (x \# xs) = f \quad x \quad \{\text{fold } f \ e \ xs\}
\]
Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

\[ \text{fold } f \ e \ (x \# \ xs) = f \]

- new element \(x\)
- old (intermediate) result \((\text{fold } f \ e \ xs)\)

Limitations
This works well with single folds.
For nested folds more powerful equations are needed.
- commutative operations
- multisets instead of lists
Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

\[
\text{fold } f \ e \ (x \# \mathbf{xs}) = f \quad \underbrace{x}_{\text{new element}} \quad \underbrace{\text{(fold } f \ e \ \mathbf{xs})}_{\text{old (intermediate) result}}
\]

Limitations

- This works well with single folds.
- For nested folds more powerful equations are needed.
Automatic Incrementalisation of Report Functions

Basic idea: unfolding folds

\[ \text{fold } f \ e \ (x \# xs) = f(x) \bigcup \bigcup_{(\text{fold } f \ e \ xs)} \]

new element

old (intermediate) result

Limitations

- This works well with single folds.
- For nested folds more powerful equations are needed.
  - commutative operations
  - multisets instead of lists
Outline

1. Enterprise Resource Planning Systems
2. Reports & Report Functions
3. Conclusions
Conclusions
The Last Slide

What do we have?

- Simple yet powerful data model for ERP
- Purely functional language for extracting & aggregating complex information
- Highly customisable & flexible
- Incrementalisation of report functions
Conclusions

The Last Slide

What do we have?

- Simple yet powerful data model for ERP
- Purely functional language for extracting & aggregating complex information
- Highly customisable & flexible
- Incrementalisation of report functions

What are we planning?

- More powerful incrementalisation transformations
- Possibly restricting the language further
- A better cost model