DSLs for Finance!

Fritz Henglein
Department of Computer Science, University of Copenhagen (DIKU)

DSL4EE Workshop, Marstrand, 2011-06-17

Outline

- DSL essentials
- DSLs in enterprise modeling
- DSLs in Finance -- observations and considerations
- HIPERFIT

What is a DSL?

In our opinion, FIDO is a compelling example of a domain-specific language. It is focused on a clearly defined and narrow domain: *formulas in monadic second-order logic* or, equivalently, *automata on large alphabets*. It offers solutions to a classical software problem: *drowning in a swamp of low-level encodings*. It advocates a simple design principle: *go by analogy to standard programming language concepts*. It uses a well-known and trusted technology: *all the phases of a standard compiler, including optimizations at all levels*. It provides unique benefits that cannot be matched by a library in a standard programming language: *notational conveniences, type checking, and global optimizations*. And during its development, we discovered new insights about the domain: *new notions of tree automata and algorithms*.

Klarlund, Schwartzbach, *A domain-specific language for regular sets of strings and trees*, IEEE Trans. Software Engineering, 3:25, 1999

DSL approach: Motivation

- Design DSLs that capture compositional structure of domain model
  - Isomorphism principle: One-to-one correspondence between informal requirements and formal DSL specifications
  - Small change in requirements = Small change in specifications
- Language-oriented programming

What is special about DSLs (for behavior)?

- A DSL specification is
  - a **program**: it has standard semantics
  - **data**: it can be analyzed
- A DSL specification has **multiple, open-ended interpretations**

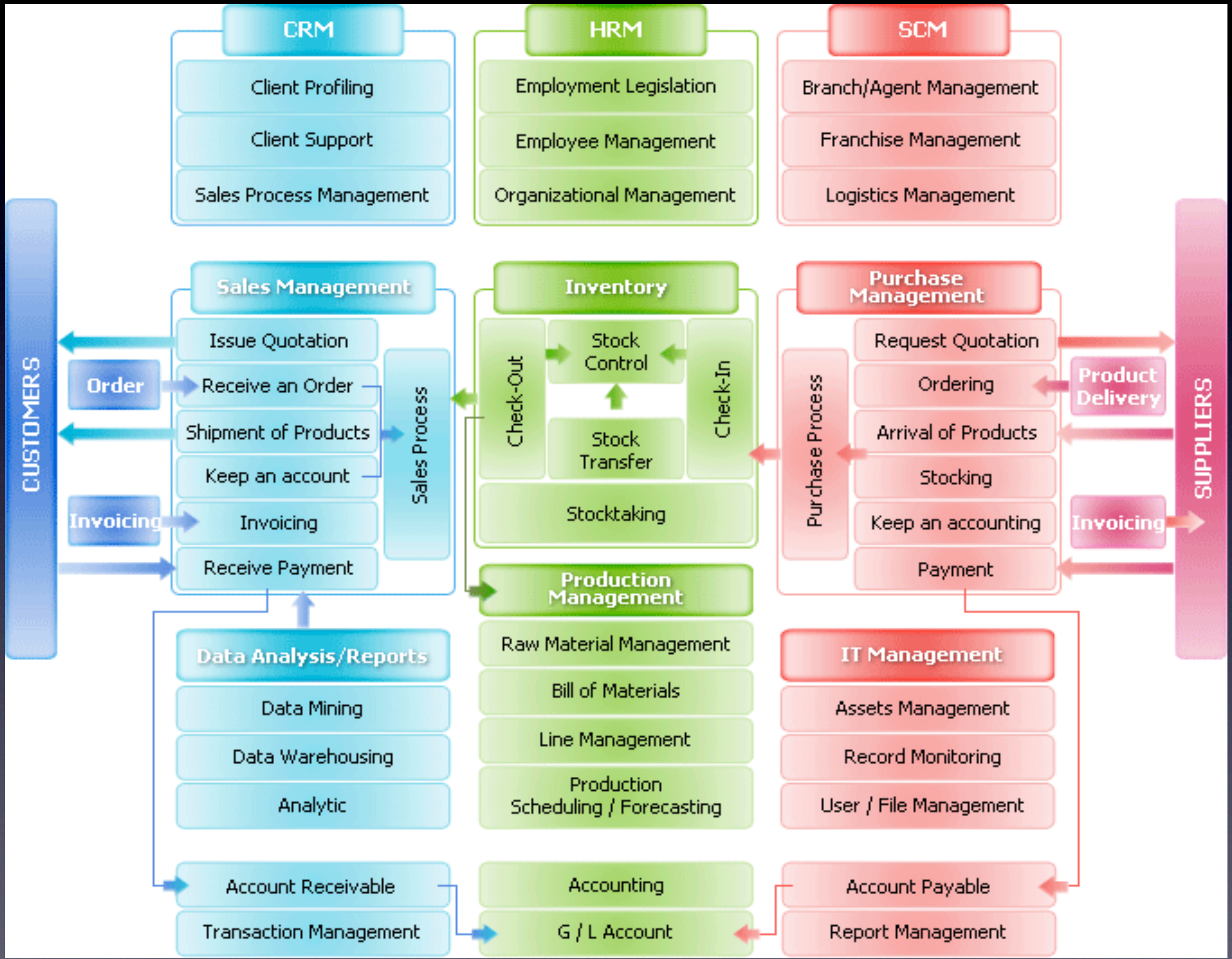
ERP System

- Electronically manage everything in a company
  - sales, purchase, production orders
  - payments
  - inventory
  - customers
- Perform analysis on business data
  - tax
  - statistics

ERP market

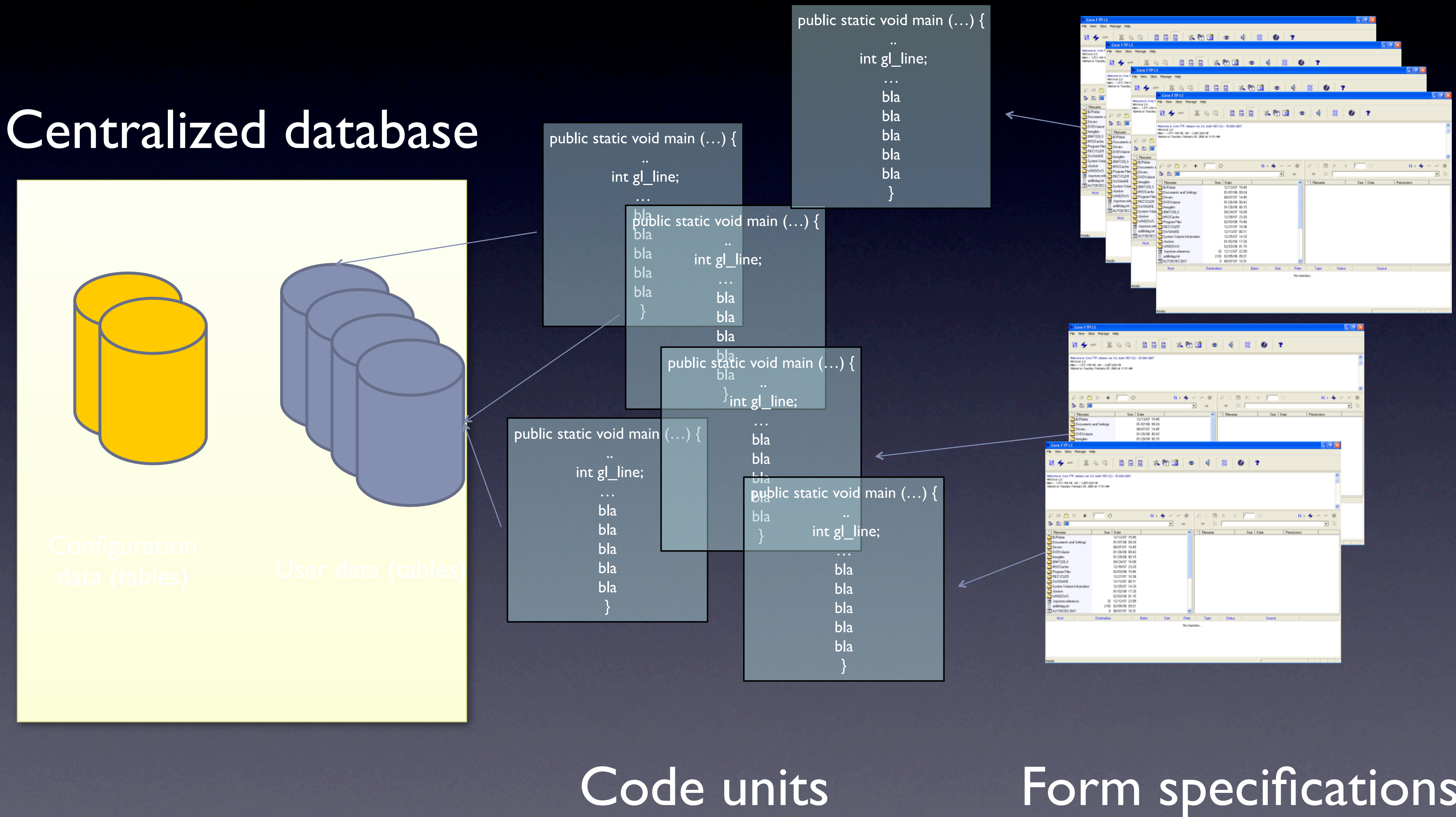
- SAP, Oracle,
- Microsoft Dynamics, many more (mostly regional)
  - Inclusive definition: "enterprise systems" (ERP, CRM, HRM, SCM, etc.)
- Global annual revenues (2009, Gartner Group projections):
- Hollywood: 28 billion dollars (2008)
- Videogames: 42 billion dollars
- ERP *Software*: 223 billion dollars

Functional View of present-day ERP System Architecture

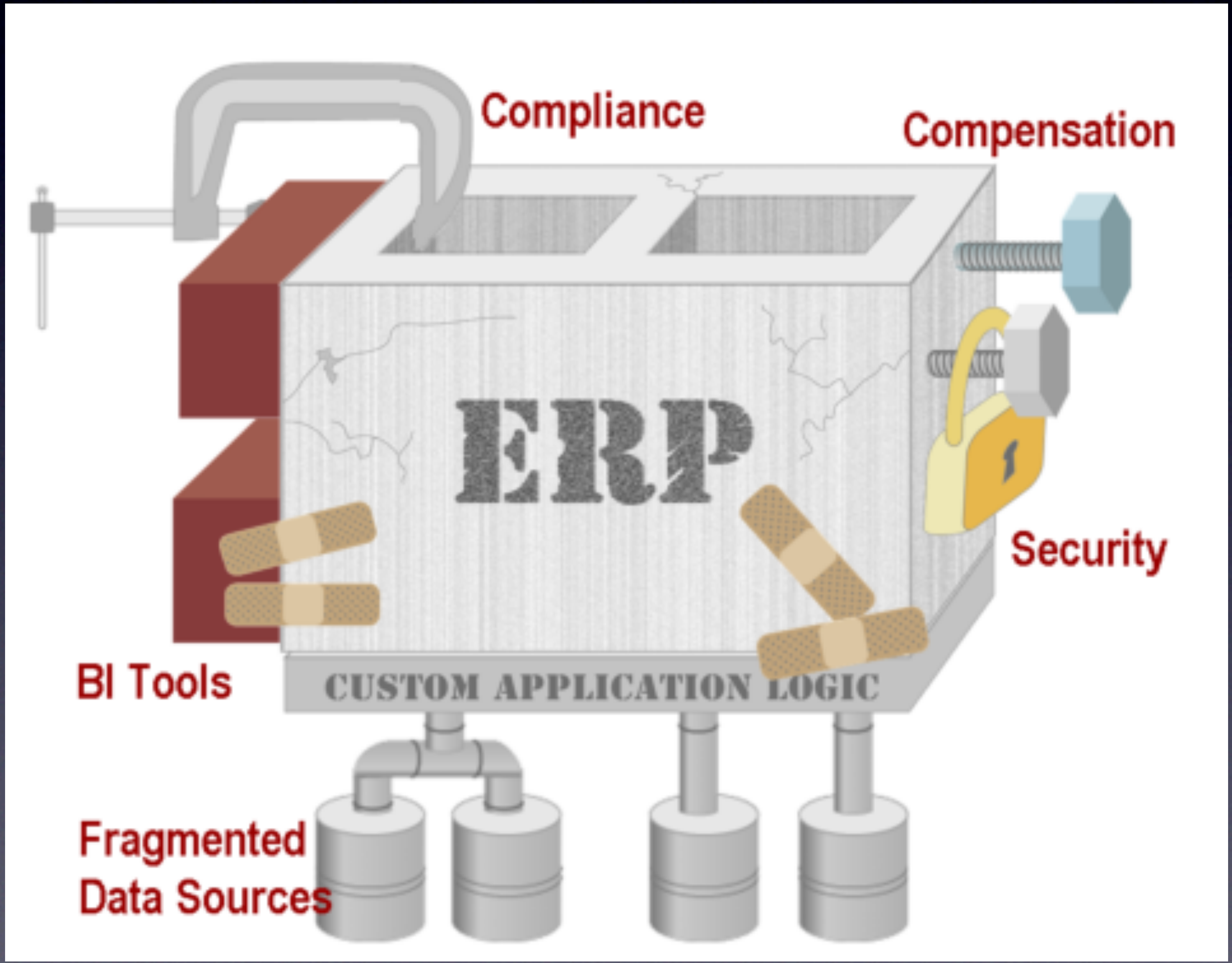


Source: http://www.bluedzine.com/images/erp_diagram.gif
Source: http://blogs.zdnet.com/SAAS/images/erp-block.png

Architectural view:
Conventional ERP software architecture



Developer's view of ERP Systems



Source: http://blogs.zdnet.com/SAAS/images/erp-block.png
Source: http://www.bluedzine.com/images/erp_diagram.gif

Process-Oriented Event-driven Transaction System (POETS): Requirements = Specifications = Code

- **Data**: Resources, events (”transactions”), agents, documents (basic information such as invoices)
- **Reports**: Interpretation of all base data by selection, aggregation, correlation, transformation etc.
- **Processes**: Specifications of *expected* sequences of events, in particular (commercial) contracts
- **Rules**: Legal and business constraints on how things are to/may be done, e.g. VAT or customs rules
- **Interfaces**: Specification of interactions between system components, and between system and users (roles).

Why POETS?

- No accounting artifacts (double-entry book-keeping): register events
- Unlimited configurability by DSLs:
  - Contract, report, rules languages
- Technical “simplicity”:
  - Order of magnitude less code
  - Performance is “in the box” (needs not be programmed)

DSLs: The business model aspect

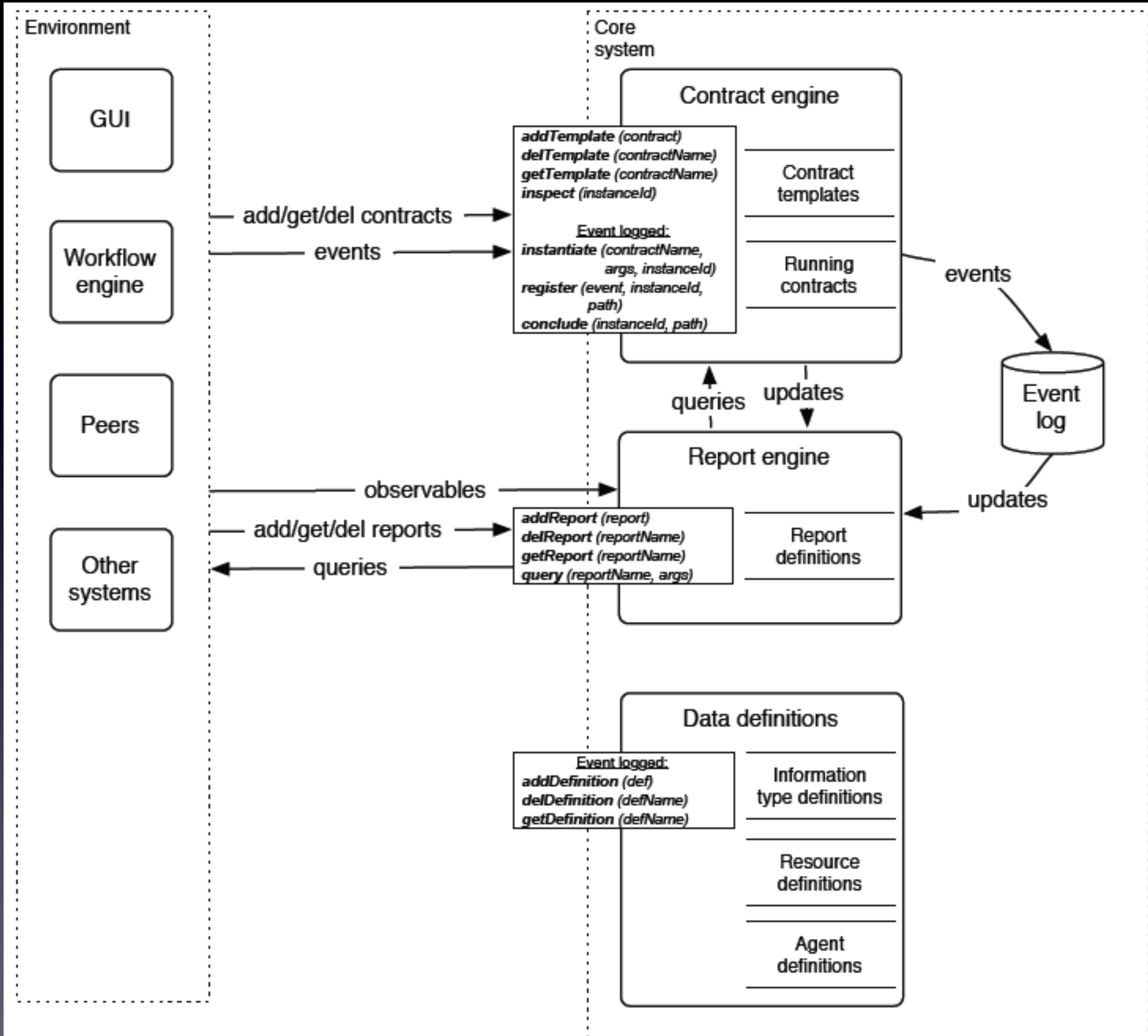
- Free clients: Android, Web, iPhone, iPad
- Free servers: Cloud-based
- Apps containing business processes, rules, information
  - Developed by channel partners
  - Made possible by DSL architecture

Unique features

- Simplicity ... and generality
  - No SQL, no legacy code, no double entry bookkeeping, no platform dependence
- Built-in auditability (like Time Machine)
- Unlimited extensibility through DSLs
  - new applications possible
  - scalability through partner channels

New apps possible in POETS (examples)

- Render contracts in Hindi
  - Works for all contracts, also future/new ones
- Sales tax rules for the State of New Hampshire, e.g. for approval by regulatory body
  - Not enmeshed with code for processing them
- Stochastic contract valuation for risk management (compare with pricing of financial instruments)



POETS Architectur
e
(more
detail)

1 Receive 3 iPhones and 2 MacBooks from supplier X

2 Receive 2 iPhones and 1 MacBooks from supplier Y

3 Receive an invoice from X for 3 iPhones (3 * 400 USD incl. VAT) and 2 MacBooks (2 * 2000 USD incl. VAT) and rush delivery charge (20 USD – VAT exempt)

4 Receive invoice from Y for 3 iPhones (3 * 420 USD incl. VAT) and 2 MacBooks (2 * 1940 USD incl. VAT) and shipping (100 USD incl. VAT)

5 Deposit 5220 USD into X's bank account

6 Send check to Y to the amount of 5240 USD

7 Observe on our bank account that check has been cashed

8 Receive order from A of 1 MacBook and 1 iPhone priced at 3000 USD incl. VAT

9 Deliver 1 MacBook and 1 iPhone to A

10 Receive from A 3000 USD into our bank account

11 Pay VAT due

12 *A year passes*

13 Deliver, invoice, and receive payment for 1 MacBook worth $800 incl. VAT to Z

Real events

1	Receive 3 iPhones and 2 MacBooks from supplier X	transmit(X, C, 3 iPhone + 2 Mac, 2008-01-15)
2	Receive 2 iPhones and 1 MacBooks from supplier Y	transmit(Y, C, 2 iPhone + 1 Mac, 2008-01-19)
3	Receive an invoice from X for 3 iPhones (3 * 400 USD incl. VAT) and 2 MacBooks (2 * 2000 USD incl. VAT) and rush delivery charge (20 USD – VAT exempt)	inform(X, C, invoice (iPhone, 3, 400 USD, 25%), (Mac, 2, 2000 USD, 25%), (fee, 1, 20 USD, 0%), 2008-01-19)
4	Receive invoice from Y for 3 iPhones (3 * 420 USD incl. VAT) and 2 MacBooks (2 * 1940 USD incl. VAT) and shipping (100 USD incl. VAT)	inform(Y, C, invoice (iPhone, 3, 1680 USD, 25%), (Mac, 2, 7760 USD, 25%), (shipping, 1, 400 USD, 25%), 2008-01-19)
5	Deposit 5220 USD into X's bank account	transmit(C.bank, X.bank, 26100 USD, 2008-01-22)
6	Send check to Y to the amount of 5240 USD	transmit(C, Y, right to draw 26.300 USD from C.bank, 2008-01-22)
7	Observe on our bank account that check has been cashed	transmit(C.bank, Y.bank, 26300 USD)

...

Registered events

...

8	Receive order from A of 1 MacBook and 1 iPhone priced at 3000 USD incl. VAT	enter contract for exchange of (1 iPhone + 1 Mac, 1, 12000 USD, 25%)
9	Deliver 1 MacBook and 1 iPhone to A	transmit (C, A, 1 Mac, 2008-01-26)
10	Receive from A 3000 USD into our bank account	transmit (A.bank, C.bank, 15000 USD 2008-01-30)
11	Pay VAT due	transmit (C.bank, IRS, VAT_due())
12	*A year passes*	
13	Deliver, invoice, and receive payment for 1 MacBook worth $800 incl. VAT to Z	transmit (C.ops, Z, 1 Mac, 2009-01-30); inform (C, Z, invoice (Mac, 1, 3200 USD, 25%), 2009-01-30); transmit (Z.bank, C.bank, 4000 USD, 2009-02-06);

Registered events

Reports: Invoices

$$
\begin{aligned}
InvoicesReceived = \{(i,(A,R,(price,t)))\\
:(inform(A,B,(R,price),t,i) \in Events \mid B \leq me, A \not\leq me\}\\
InvoicesSent = \{(i,(A,R,(price,t)))\\
:(inform(A,B,(R,price),t,i) \in Events \mid A \leq me, B \not\leq me\}
\end{aligned}
$$

Reports: FIFO inventory valuation for cost

foldl: Iterate over inventory acquisitions from oldest to youngest.
Replace by foldr ➔ LIFO costing

Invoices sent

$$GoodsSold = \sum \{R : (i, (A, R, price)) \in InvoicesSent\}$$

$$FIFOCost = foldl(accumCost, (0, GoodsSold), InvAcq)$$

Inventory acquired

where

$$accumCost((R, (p, m, t)), (total, Q)) =$$

$$\text{let } (R', Q') = Subtract(R, Q) \text{ in}$$

$$(total + p(R - R'), Q')$$

$$\text{end}$$

Contracts: Processes involving external parties

Choreography ("global view")

A simple sales contract, with VAT requirements

```
Sale (vendor, customer, resource, pinfo as (p, m), deadline) =
  transmit (vendor, customer, resource, T | T <= deadline) ||
  (inform (vendor, customer, (resource, pinfo), T').
    (transmit (Tax, vendor, -m(resource) DKK, _ ) ||
     transmit (customer, vendor, (p + m)(resource) DKK, T''
               | T'' <= T' + 8 days)))
```

Contracts…

A sales contract with multiple installments (and VAT payments)

```
Sale (vendor, customer, resource, pinfo as (p, m), deadline) =
   transmit (vendor, customer, resource, T | T <= deadline) ||
   (inform (vendor, customer, (resource, pinfo), T').
     (transmit (Tax, vendor, -m(resource) DKK, _ ) ||
      TransmitM (customer, vendor, (p + m)(resource) DKK,
                 T' + 8 days)
```

where

```
TransmitM (customer, vendor, resource, deadline) =
   (transmit (customer, vendor, R, T |
              T <= deadline, 0 < R <= resource).
    TransmitM (customer, vendor, resource - R, deadline))
   Success
```

Why formal contract specifications?

- Operational semantics by reduction semantics
  - <C, e> → C' (e matches, C' is residual contract)
  - <C, e> → "unexpected event"
- Can write programs that analyze contracts C:
  - When is the next deadline for something to happen in C?
  - Is the next thing to happen a payment I must make? (A/P)
  - Is the next thing to happen a payment I must receive? (A/R)
  - What is the value/risk to me of entering into C? (Peyton-

Internal processes

The universal process, which allows all internal transactions, for matching events that are not part of a given contract or specified internal business process

```
UniversalProcess() =
 ( transmit (A, B, R, T | A <= Me, B <= Me) +
   inform (A, B, info, T | A <= Me, B <= Me) +
   transform (A, R1, R2, T | A <= Me)
 );
   UniversalProcess()
```

Contract Specification Language (CSL), v. 2.0

$$
\begin{array}{lll}
s ::= \textbf{letrec } \{f_i(\vec{x}_i)\langle\vec{y}_i\rangle = c_i\}_{i=1}^{n} \textbf{ in } c \textbf{ starting } e & & \text{(CSL specification)} \\
\\
c ::= \textbf{fulfillment} & & \text{(No obligations)} \\
\quad \mid \langle e_1\rangle\ k(\vec{x}) \textbf{ where } e_2 \textbf{ due } d \textbf{ remaining } z \textbf{ then } c & & \text{(Obligation)} \\
\quad \mid \textbf{if } k(\vec{x}) \textbf{ where } e \textbf{ due } d \textbf{ remaining } z \textbf{ then } c_1 \textbf{ else } c_2 & & \text{(External choice)} \\
\quad \mid \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2 & & \text{(Internal choice)} \\
\quad \mid c_1 \textbf{ and } c_2 & & \text{(Conjunction)} \\
\quad \mid c_1 \textbf{ or } c_2 & & \text{(Disjunction)} \\
\quad \mid f(\vec{e}_1)\langle\vec{e}_2\rangle & & \text{(Instantiation)} \\
\\
e ::= x \mid v \mid \neg e \mid e_1 \star e_2 \mid e_1 \prec e_2 & & \text{(Expression)} \\
d ::= \textbf{after } e_1 \textbf{ within } e_2 & & \text{(Deadline expression)}
\end{array}
$$

Figure 2: The grammar of CSL. $f \in \mathcal{F}$ ranges over template names, $x, y, z \in \mathcal{V}$ range over variables, $k \in \mathcal{K}$ ranges over action kinds, and $v \in \bigcup_{t \in \mathcal{T}} [\![t]\!]$ ranges over values. Furthermore, $\star \in \{+, -, *, /, \wedge\}$ and $\prec \in \{<, =\}$.

Hvitved, Klaedtke, Zalinescu, *A trace-based model for multiparty contracts*, to appear in JLAP 2011

Reporting In ERP Systems

- ERP systems contain a lot of data
- Many reports are simple
- Computing reports is time consuming
  - Only suitable for off-line reporting (batch runs)
  - Decisions may be delayed due to report computation time
- Reports are usually expressed in SQL, in a general purpose languages, (for instance, X++ or C/AL) or a combination of them

FunSETL: Purpose

- Powerful enough to express reports.
- Restrictive enough to facilitate automatic incrementalization.
- Easy to define report declaratively:
  - *What* should be the result?
  - Not: *How* exactly should it be computed?

FunSETL: Properties

- A simple functional language
- No recursion - only iteration on multisets
- Strongly normalizing
  - Data iteration
  - No general recursion

FunSETL: Syntax

$$
\begin{array}{rcl}
\tau & ::= & \mathbf{bool} \mid \mathbf{int} \mid \mathbf{real} \mid \mathbf{date} \mid \tau_1 + \tau_2 \mid \{lab_1 : \tau_1, \ldots, lab_k : \tau_k\} \mid \\
 & & \mathbf{map}(\tau_1, \tau_2) \mid \mathbf{mset}(\tau) \\
c & ::= & n \mid r \mid \textit{yyyy-mm-dd} \mid \mathbf{true} \mid \mathbf{false} \\
binop & ::= & + \mid - \mid * \mid / \mid = \mid <= \mid < \mid \mathbf{and} \mid \mathbf{or} \mid \\
 & & \mathbf{with} \mid \mathbf{inter} \mid \mathbf{union} \mid \mathbf{diff} \mid \mathbf{in} \mid \mathbf{subset} \\
unop & ::= & \mathbf{not} \mid \mathbf{dom} \\
e & ::= & x \mid e_1 \; binop \; e_2 \mid unop \; e \mid \mathbf{inL}(e) \; \mathbf{as} \; \tau \mid \mathbf{inR}(e) \; \mathbf{as} \; \tau \mid \\
 & & \mathbf{valL}(e) \mid \mathbf{valR}(e) \mid \{lab_1 := e_1, \ldots, lab_k := e_k\} \mid \#lab(e) \mid \\
 & & f(e_1, \ldots, e_m) \mid [] \; \mathbf{as} \; \tau \mid e[e'] \mid e[e_1 \rightarrow e'_1] \mid \{\} \; \mathbf{as} \; \tau \mid \\
 & & \mathbf{if} \; e_1 \; \mathbf{then} \; e_2 \; \mathbf{else} \; e_3 \mid \mathbf{foreach} \; (a, b \rightarrow e_1) \; e_2 \; e_3 \mid \\
 & & \mathbf{let} \; x = e_1 \; \mathbf{in} \; e_2 \\
fdecl & ::= & \mathbf{fun} \; id(x_1 : t_1, \ldots, x_m : t_m) = e
\end{array}
$$

Incrementalization

- Idea: Reuse intermediate results automatically.
- Let f be a function and $\oplus$ be an update operation.
- Then f' is the incremental version of f with respect to $\oplus$
- It computes the result of $f\,(x \oplus y)$ by making use of the value of $f\,(x)$:

The key equivalence

- We are interested in incrementalization with respect to the **with** operation.
- That is, we want to eliminate **foreach** loops by the following equivalence.
  - Realizes asymptotic speed-up over naïve execution

- *r* = ***foreach*** *(a, b* → *e1) e2 S* implies

Transformations

- To transform function into an incrementalized version perform the following transformations:
  - 1. Normalise to A-normal form (almost)
  - 2. Caching of intermediate results
  - 3. Incrementalization use cashed intermediate values
  - 4. Prune all unnecessary computations

Example: Computing the Average

```
fun count(s : mset(int)) = foreach (x, sum => sum + 1) 0 s

fun average(s : mset(int)) =
   foreach (x, sum => x + sum) 0 s/count(s)
```

Example: Normalisation

```
fun  count(s : mset(int)) =
    let tmp1 = foreach (x, sum => sum + 1) 0 s in
    tmp1

fun  average(s : mset(int)) =
    let tmp2 = foreach (x, sum => x + sum) 0 s in
    let tmp3 = count(s) in
    let tmp4 = tmp2/tmp3 in
    tmp4
```

Example: Cache Intermediate Results

```
fun  count(s : mset(int)) =
    let tmp1 = foreach (x, sum => sum + 1) 0 s in
    {1 = tmp1; 2 = tmp1}

fun  average(s : mset(int)) =
    let tmp2 = foreach (x, sum => x + sum) 0 s in
    let tmp3 = count(s) in
    let tmp4 = {1 = tmp2/#1(tmp3)} in
    {1 = #1(tmp4); 2 = tmp2; 3 = tmp3; 4 = tmp4}
```

Example: Incrementalization

```
type count_rt = {1 : int; 2 : int}
fun  count(s : mset(int), e : int, r : count_rt) =
    let tmp1 = #2(r) + 1 in
    {1 = tmp1; 2 = tmp1}

type average_rt = {1 : int; 2 : int; 3 : count_rt; 4 : int}
fun  average(s : mset(int), e : int, r : average_rt) =
    let tmp2 = e + #2(r) in
    let tmp3 = count(s, e, #3(r)) in
    let tmp4 = {1 = tmp2/#1(tmp3)} in
    {1 = #1(tmp4); 2 = tmp2; 3 = tmp3; 4 = tmp4}
```

Example: Cleaning Up

```
type count_rt = {1 : int; 2 : int}
fun count(r : count_rt) =
   let tmp1 = #2(r) + 1 in
   {1 = tmp1; 2 = tmp1}

type average_rt = {1 : int; 2 : int; 3 : count_rt}
fun average(e : int, r : average_rt) =
   let tmp2 = e + #2(r) in
   let tmp3 = count(#3(r)) in
   let tmp4 = tmp2/#1(tmp3) in
   {1 = tmp4; 2 = tmp2; 3 = tmp3}
```

Case study

- Financial statement (report) in MS Dynamics AX
- X++ source of report and live data from German company
  - Provided by MDCC
- Computes:
  - Sum class computations (balance of account intervals X000 − X999, where X = 0, 1, 2, 3, 4, 5, 6, 7, 8 and from 9000 and up.
  - Assets and liabilities

Status

- Incrementalizer and compiler in F#
  - Target language: C#
  - C# then compiled to .NET (MSIL)
- Interrailing with MS Dynamics NAV
  - FunSETL report can be executed and displayed in NAV (Version 5.0).

Challenges for innovation in ERP sector

- Technical backwards compatibility:
  - Vast amounts of code and data representing enormous investments
  - Investment in code and data must be preserved
- Eco-sociological backwards compatibility:
  - Large number of existing users, business architects, and (non-CS trained) application developers

Domain-specific languages: Advantages

- Domain-oriented concepts
  - Reflects domain ontology: Resource, Event, Agent, etc.
- Built-in properties
  - Checkpointing: Source-level representation of execution state
  - Termination/limited resource usage
- Restrictive expressive power
  - Analyzability, not just executability
- Closure properties by composition
  - Differentiability

Financial contracts: Steps

Contract	American put option
Execution strategy	Exercise option if price good and counterparty risk low
Analysis (for given scenario)	Computed pay-off of a partiular future contract execution scenario, discounted to today
(stochastic) model	Underlying modeled using Brownian motion, with given mean and variance
Analysis (for model)	Expected pay-off and its variation (and, implicitly, execution strategy)

DSLs?

Contract	often implicit, or enumerated (not defined)
Execution strategy	no
Analysis (for given scenario)	no
(stochastic) model	no (programmatic generation of scenarios)
Analysis (for model)	PDEs, closed forms (where possible) , no where not

Why DSLs for Finance?

- Multiple independent use of what is modeled
  - Contracts: For lawyers, execution
  - Stochastic models: For multiple contracts
  - Analysis functions: For arbitrary contracts, arbitrary models
  - Solution methods: MC, FD, etc.
- Genericity: infinitely many specifications, “intensional” representation -> functions that work on infinitely many different specifications

DSL benefits

- **DSLs for correctness, safety and reusability**
  - Click and run and do your job
  - Language invariants, properties, logic
- **DSLs for expressiveness, performance and business scalability**
  - Unlimited extensibility
  - Cutting-edge “computer science in the box”
  - Partner business model through DSL (micro-)app market
- **DSLs for separation of concerns and standardization**
  - Include in legal rules

Exercise

- Design a DSL for waterfall payments part of asset-backed securities

SECURITIES AND EXCHANGE COMMISSION

17 CFR Parts 200, 229, 230, 232, 239, 240, 243 and 249

Release Nos. 33-9117; 34-61858; File No. S7-08-10

RIN 3235-AK37

ASSET-BACKED SECURITIES

AGENCY: Securities and Exchange Commission

ACTION: Proposed rule.

SUMMARY: We are proposing significant revisions to Regulation AB and other rules regarding the offering process, disclosure and reporting for asset-backed securities. Our

format using eXtensible Markup Language (XML). In addition, we are proposing to require, along with the prospectus filing, the filing of a computer program of the contractual cash flow provisions expressed as downloadable source code in Python, a

1

A programming language!