Domain-Specific Languages for Enterprise Systems

Jesper Andersen[2] Patrick Bahr[1]
Fritz Henglein[1] Tom Hvitved[1]

[1]University of Copenhagen, Department of Computer Science

[2]Configit A/S, Copenhagen

ISoLA '14, 8th October, 2014

Enterprise Resource Planning (ERP) Systems

Goal: integrate several software components that are essential for managing a business.

Enterprise Resource Planning (ERP) Systems

Goal: 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
- ...

Traditional ERP Systems

Three tier architecture

- client
- application server
- relational database

Traditional ERP Systems

Three tier architecture

- client
- application server
- relational database

Shortcomings

- database combines transactional data & implicit process state
- processes are implemented in general purpose language
- semantic gap between specification and implementation
- large monolithic systems
- hard to maintain

Entering POETS

Process-oriented event-driven transaction systems

Entering POETS

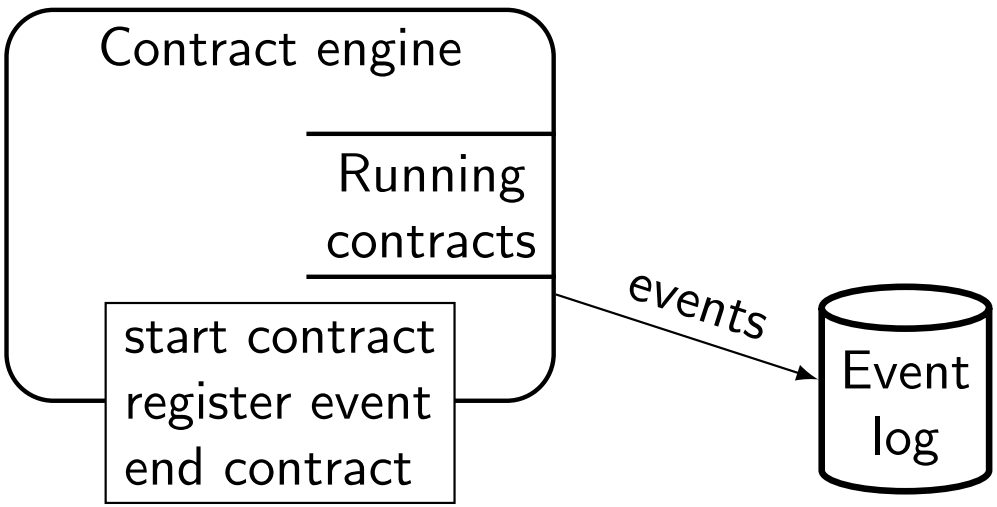
Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model

Entering POETS

Process-oriented event-driven transaction systems

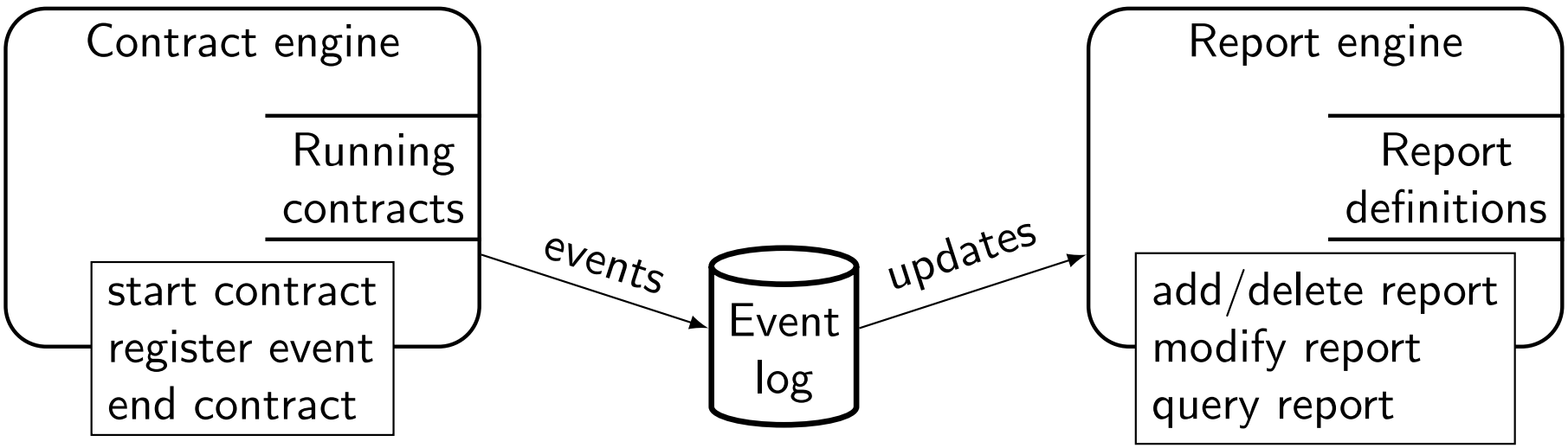
compact core system • customisable via DSLs • simple data model



Entering POETS

Process-oriented event-driven transaction systems

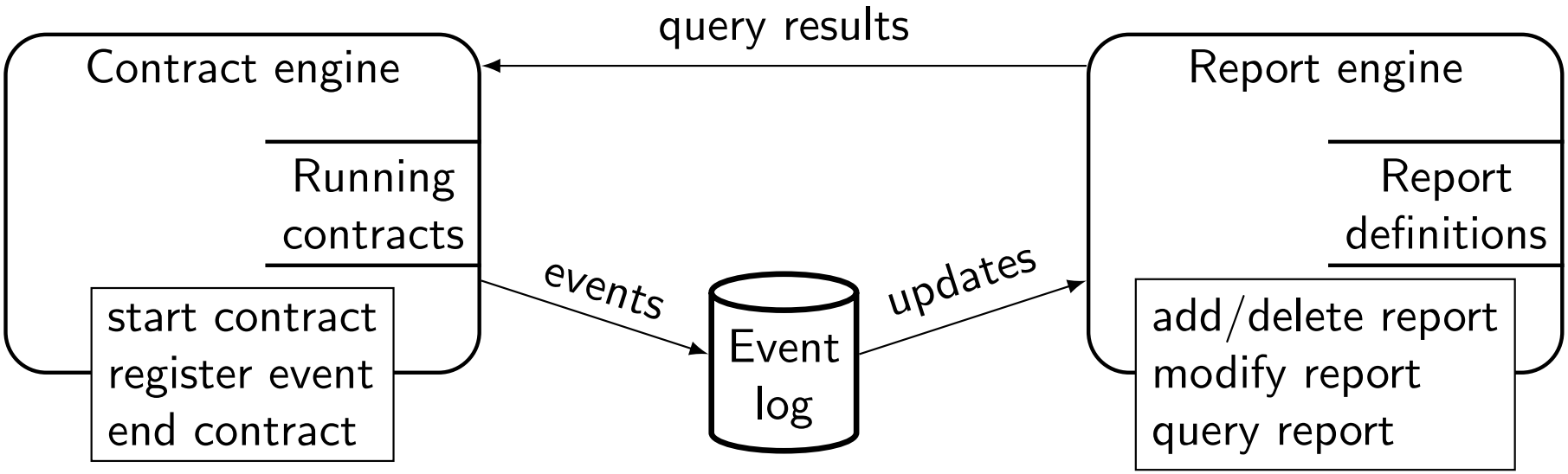
compact core system • customisable via DSLs • simple data model



Entering POETS

Process-oriented event-driven transaction systems

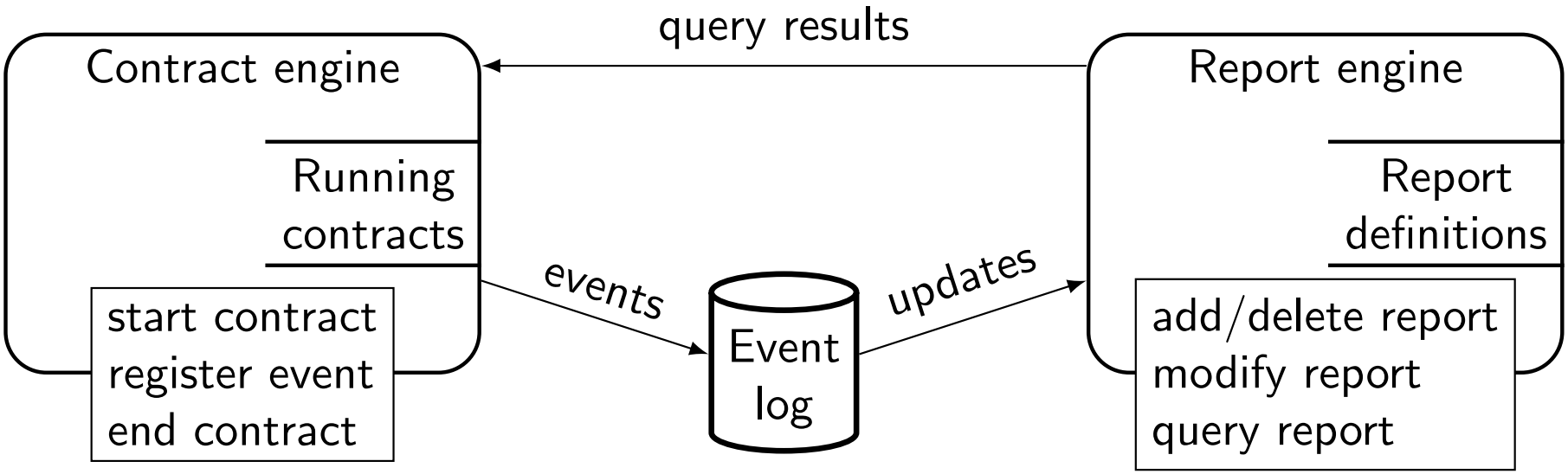
compact core system • customisable via DSLs • simple data model



Entering POETS

Process-oriented event-driven transaction systems

compact core system • customisable via DSLs • simple data model



Goal: POETS reflects the ontological architecture for requirements

The Language of POETS

Examples and Demo

1. Ontology language
2. Contract language
3. Report language

The Language of POETS

Examples and Demo

1. Ontology language — data model
2. Contract language — business processes
3. Report language — high-level data

The Language of POETS

Examples and Demo

1. Ontology language — data model
2. Contract language — business processes
3. Report language — high-level data

Example: a bike shop

Ontology of a Bike Shop

OrderLine is Data.
OrderLine has an Item.
OrderLine has Money called unitPrice.
OrderLine has a Real called vatPercentage.

Item is a Resource.
Item has an ItemType.
Item has a Real called quantity.

Bicycle is an ItemType.
Bicycle has a String called model.

The Process of Selling a Bike

clause *sale*(*lines* : [OrderLine])⟨*me* : ⟨Me⟩, *customer* : ⟨Customer⟩⟩ =
⟨*me*⟩ IssueInvoice(*sender s*, *receiver r*, *orderLines sl*)
where $s \equiv me \wedge r \equiv customer \wedge sl \equiv lines \wedge inStock\ lines$
due within $1H$
then
payment(*lines*, *me*, $10m$)⟨*customer*⟩
and
⟨*me*⟩ Delivery(*sender s*, *receiver r*, *items i*)
where $s \equiv me \wedge r \equiv customer \wedge i \equiv map\ (\lambda x \rightarrow x.item)\ lines$
due within $1W$

Demo

Adding a Repair Service

clause *sale*(*lines* : [OrderLine])⟨*me* : ⟨Me⟩, *customer* : ⟨Customer⟩⟩ =
⟨*me*⟩ IssueInvoice(*sender s*, *receiver r*, *orderLines sl*)
where $s \equiv me \wedge r \equiv customer \wedge sl \equiv lines \wedge inStock\ lines$
due within $1H$
then
payment(*lines*, *me*, $10m$)⟨*customer*⟩
and
⟨*me*⟩ Delivery(*sender s*, *receiver r*, *items i*)
where $s \equiv me \wedge r \equiv customer \wedge i \equiv map\ (\lambda x \rightarrow x.item)\ lines$
due within $1W$

Adding a Repair Service

clause *sale*(*lines* : [OrderLine])⟨*me* : ⟨Me⟩, *customer* : ⟨Customer⟩⟩ =
⟨*me*⟩ IssueInvoice(*sender s*, *receiver r*, *orderLines sl*)
where $s \equiv me \wedge r \equiv customer \wedge sl \equiv lines \wedge inStock\ lines$
due within $1H$
then
payment(*lines*, *me*, $10m$)⟨*customer*⟩
and
⟨*me*⟩ Delivery(*sender s*, *receiver r*, *items i*)
where $s \equiv me \wedge r \equiv customer \wedge i \equiv map\ (\lambda x \rightarrow x.item)\ lines$
due within $1W$
then
repair($map\ (\lambda x \rightarrow x.item)$ *lines*, *customer*, $3M$)⟨*me*⟩

Adding a Repair Service (cont.)

clause *repair*(*items* : [Item], *customer* : ⟨Customer⟩,
deadline : **Duration**)⟨*me* : ⟨Me⟩⟩ =
when RequestRepair(*sender s*, *receiver r*, *items i*)
where $s \equiv customer \wedge r \equiv me \wedge subset\ i\ items$
due within *deadline*
remaining *newDeadline*
then
⟨*me*⟩ Repair(*sender s*, *receiver r*, *items its*)
where $s \equiv me \wedge r \equiv customer \wedge i \equiv its$
due within $5D$
remaining *newDeadline*'
then
repair(*items*, *customer*, *newDeadline* $\langle - \rangle$ $5D$ $\langle + \rangle$ *newDeadline*')⟨*me*⟩

Demo

Reports

```
report : CashFlowStatement
report = let
    payments = [payment | payment : Payment ← transactions]
    mRevenues = [payment | payment ← payments, isMe (payment.receiver)]
    mExpenses = [payment | payment ← payments, isMe (payment.sender)]
  in
  CashFlowStatement{
    revenues = mRevenues,
    expenses = mExpenses,
    revenueTotal = sumPayments mRevenues,
    expenseTotal = sumPayments mExpenses}

transactions : [Transaction]
transactions = [tr.transaction | tr : TransactionEvent ← events]
```

Demo

Implementation

- server & DSLs implemented in Haskell
- client software for Android
- case studies

complete source code & documentation available online:

`https://bitbucket.org/jespera/poets/`

Contributions

- database = log + reports
- multiparty contracts with real-time constraints
- full recoverability and auditability (data and specification)
- safe run-time changes of data model, contracts and reports

Domain-Specific Languages for Enterprise Systems

Jesper Andersen[2] Patrick Bahr[1]
Fritz Henglein[1] Tom Hvitved[1]

[1]University of Copenhagen, Department of Computer Science

[2]Configit A/S, Copenhagen

ISoLA '14, 8th October, 2014

The Complete Picture