Data Constellation

Agile Information Management and Design

Semantic Modeling

A query language for the 21st century

Clifford Heath

Semantics

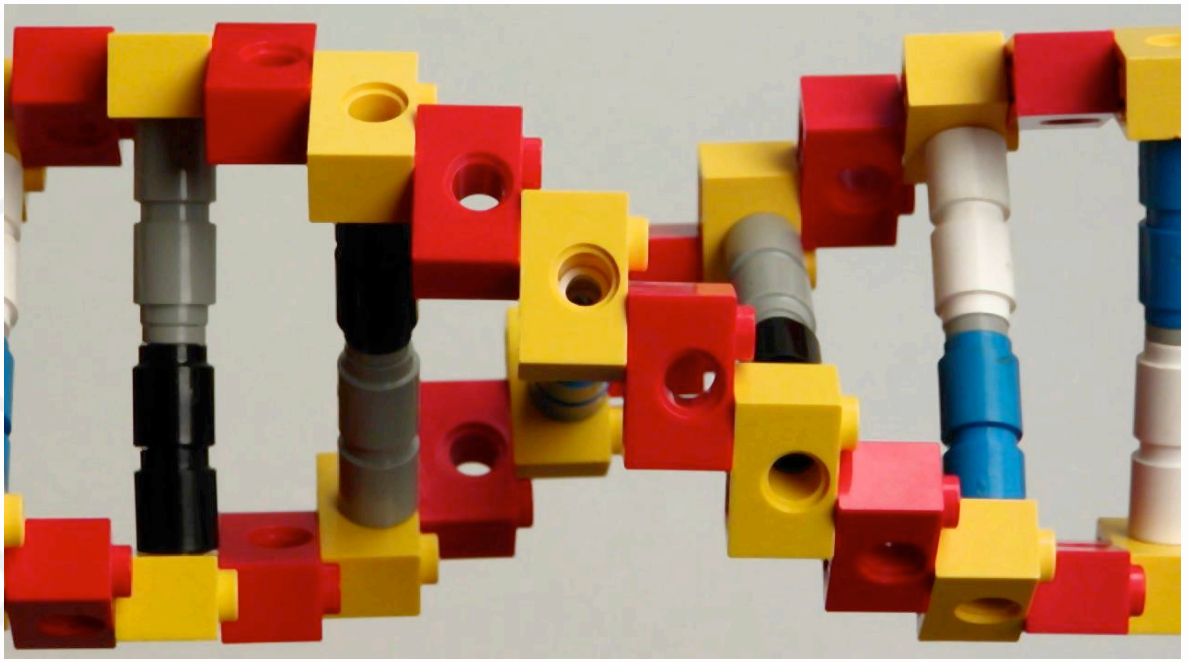
- The shared meanings of a community
- Expressed in facts

Here, a model is:

- Standard for comparison
- Simplified representation of the real world
- NOT a prototype or mockup

Communicate!

Models **must** be expressed in a way that allows all parties to understand and contribute to them



Flickr, mknowles

ActiveFacts project

A project of Data Constellation

- Constellation Query Language
- Constellation API
- GUI design tools

Elementary Facts are simple

e.g. This person is called ‘John’

The elementary form must be our code

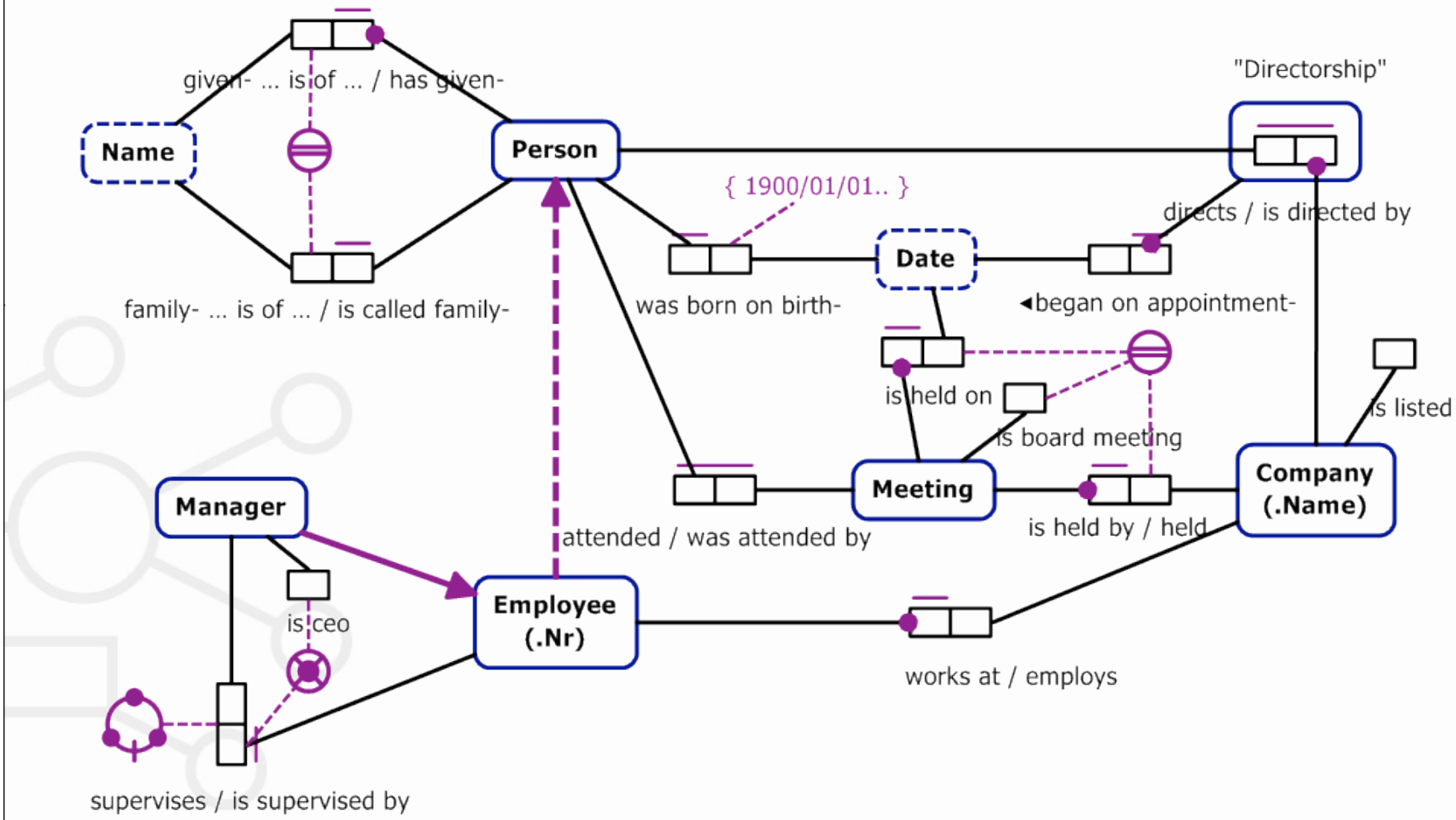
- Models must be executable
- Tools must handle the ramifications

ActiveFacts Generators

- CQL or ORM2 input
- Ruby, SQL output
- Later, C#, Java, etc, or write your own

All code you will see in this part is as generated

ORM2 Example



This and all other ORM2 examples were created using NORMA, the Neumont ORM2 Architect

CQL and ORM2 side by side

Value Types

ORM2:


```
CompanyName is written as VariableLengthText(48);
EmployeeNr is written as SignedInteger(48);
Course is written as VariableLengthText(2)
    restricted to {'A'..'E', 'PW'};
```

Each statement creates or implies two ValueTypes

Entity Types

ORM2:

CQL:

```
Company is identified by its Name;
```

=

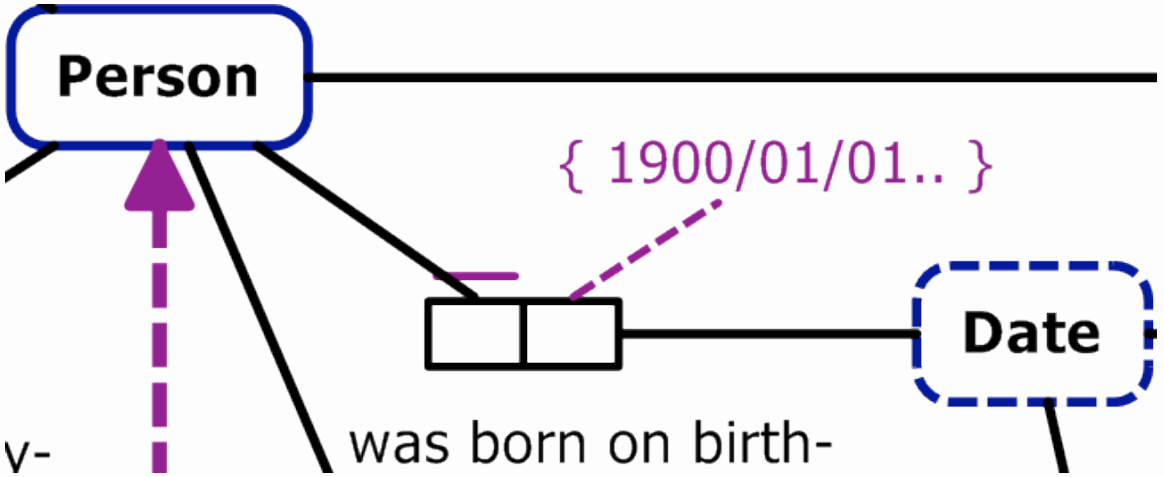
```
CompanyName is written as Name();
Company is identified by CompanyName where
        Company has one CompanyName,
        CompanyName is of at most one Company;
```

Custom reading:

```
Company is identified by its Name where
        Company is called CompanyName;
```

Fact Types

ORM2:



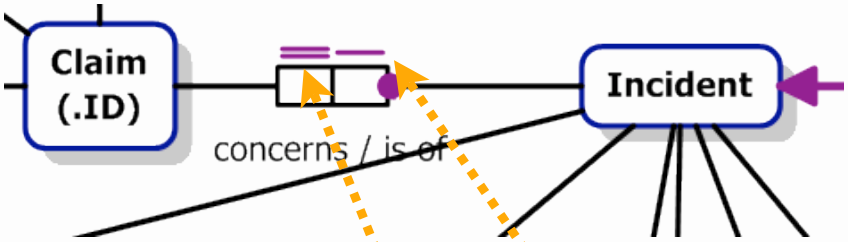
CQL:

```
Person was born on at most one birth-Date;
```

```
Person was born at at most one birth-Place;
```

```
Meeting is board meeting;
```

One-to-one



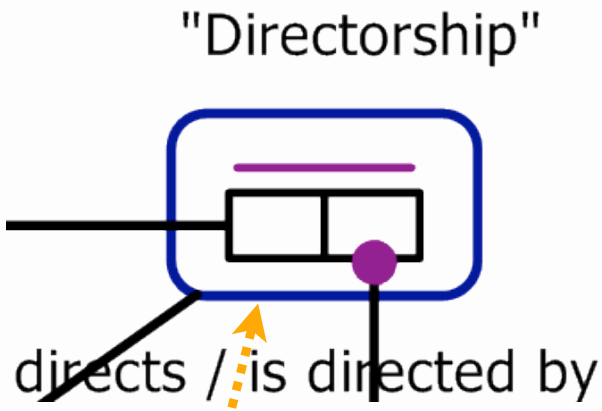

```
Claim concerns at most one Incident,
      Incident is of one Claim;
```

or more accurately:

```
Incident is identified by Claim where
      Claim concerns at most one Incident,
      Incident is of one Claim;
```

Objectified Fact Types

ORM2:



CQL:

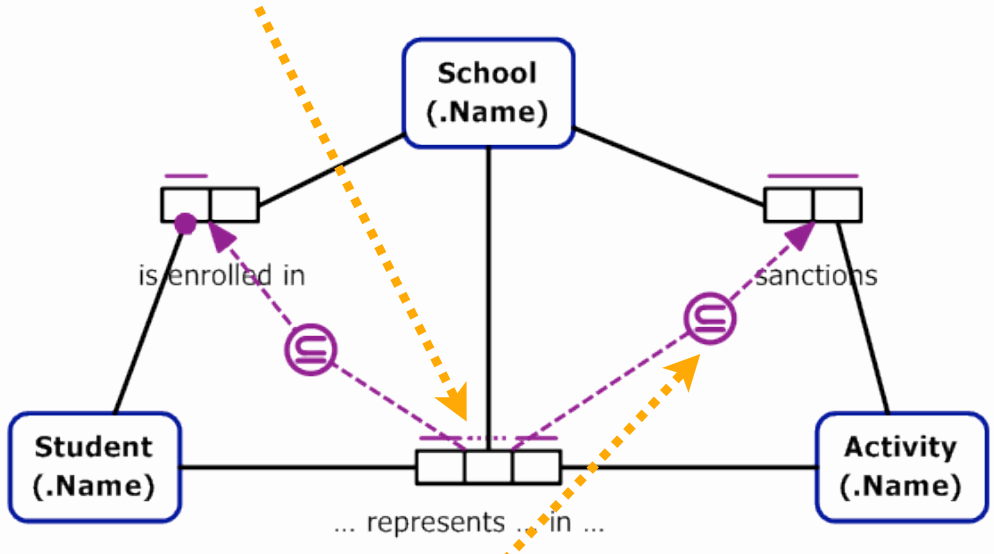
```
Directorship is where
        Person (as Director) directs Company,
        Company is directed by at least one Director;
```

and then we can:

```
Directorship began on one appointment-Date;
```

Ternary and higher

```
StudentParticipation is where
        Student represents School in Activity,
        Student participates in Activity
            which is sanctioned by at most one School;
```

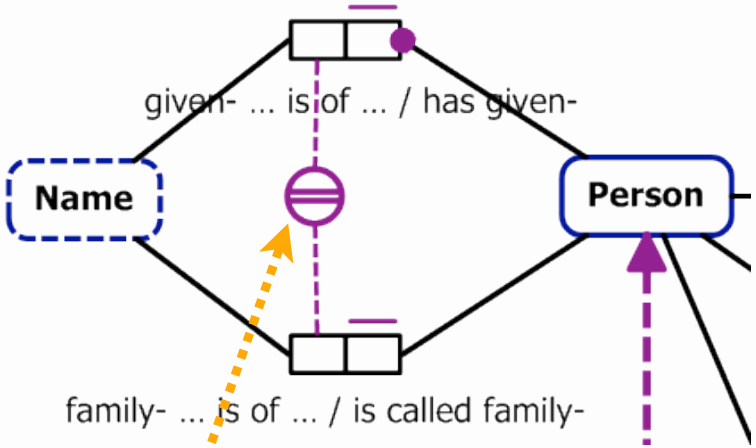


subset:

```
Student represents School in Activity
        only if School sanctions Activity;
```

Composite Identification

ORM2:



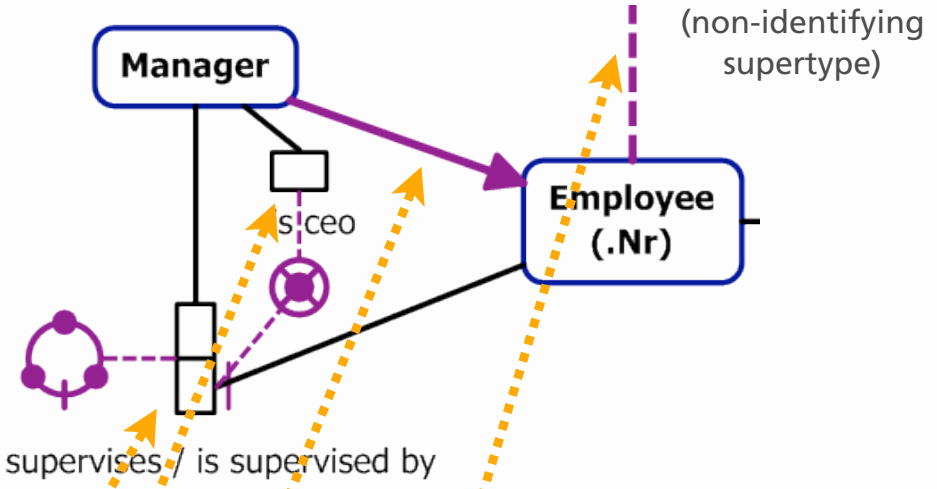
CQL:

```
Person is identified by given-Name and family-Name where
        Person has one given-Name,
        given-Name is of Person,
        family-Name is of Person,
        Person is called at most one family-Name;
```

...Two Fact Types, each with 2 readings!

Subtypes, etc

ORM2:



CQL:

```
Employee is a kind of Person identified by its Nr;
Manager is a kind of Employee;
Employee is supervised by at most one Manager [acyclic];
Manager is ceo;
```

also: unary fact type, ring constraint, XOR constraint

Fact Instances

Value Type:

Entity Type:

Composite:

```
Name 'Microsoft';

Company 'Google';

family Name'Smith' is of Person,
       Person is called given Name 'Fred';
```

... note the join over Person

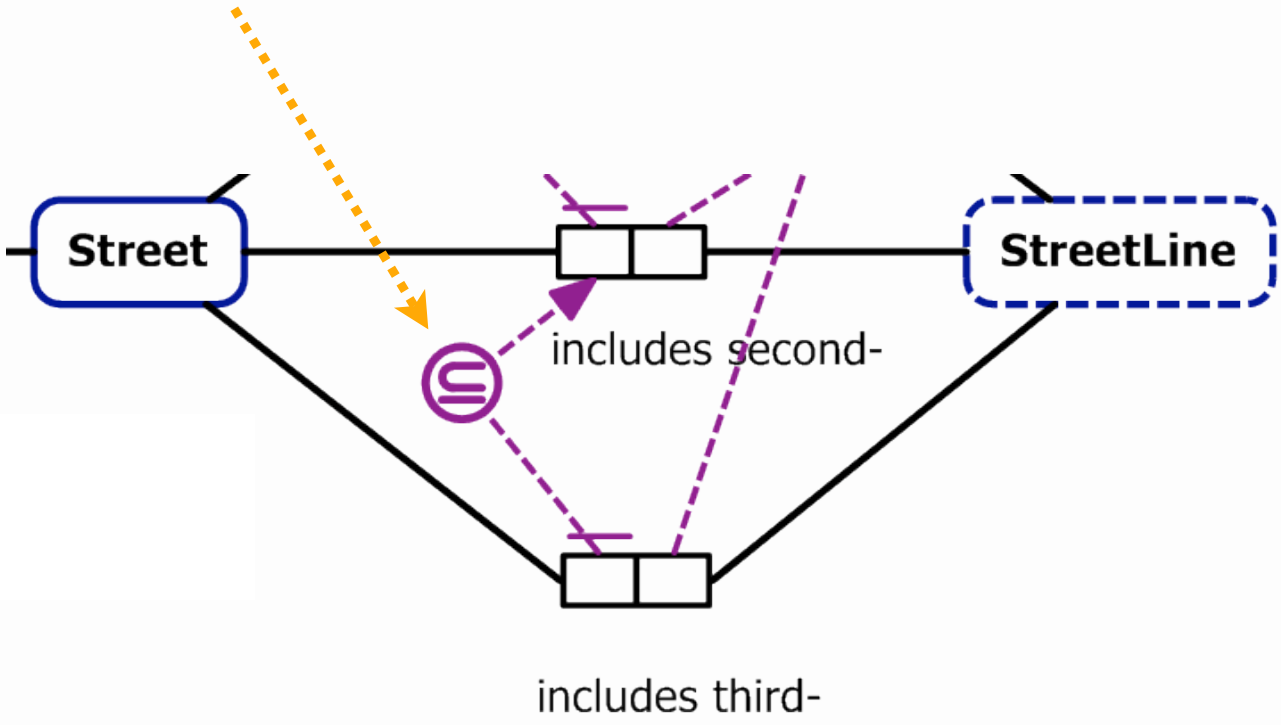
Join Contraction:

```
family Name'Smith' is of Person
       who is called given Name 'Fred';
```

... the contracted form isn't implemented yet!

Subset constraints

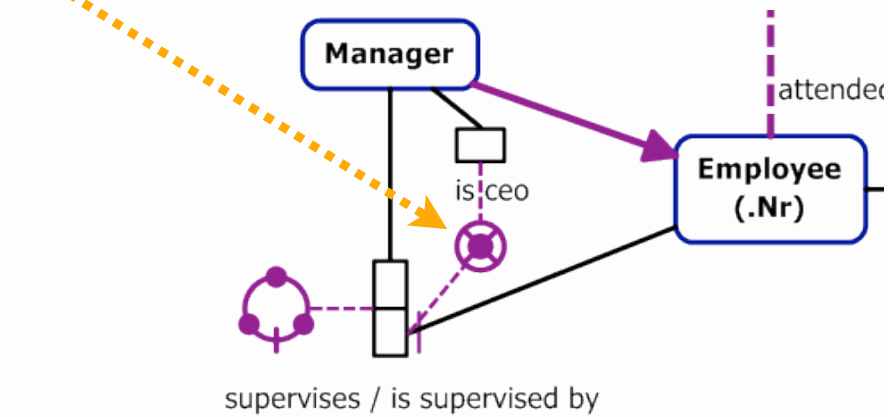
```
Street includes third-StreetLine
        only if
        Street includes second-StreetLine;
```



Mandatory / Exclusive

```
for each Employee exactly one of these holds:
        that Employee is ceo,
        that Employee is supervised by some Manager;
```

Mandatory and exclusive

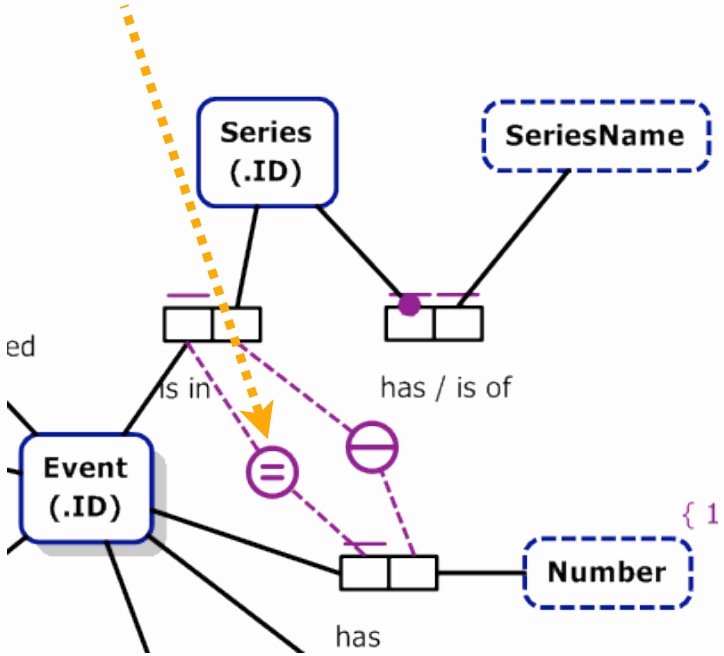


Also mandatory non-exclusive,

Exclusive non-mandatory

Equivalence

```
Event is in Series
        if and only if
        Event has Number;
```



Join constraints

```
PurchaseOrderItem matches SalesOrderItem
        only if PurchaseOrderItem is for Product
        and SalesOrderItem is for Product;
```

"Product" is the same instance in both clauses

Code Generation

From ORM to CQL

```
Company is identified by its Name where
        Company is called CompanyName;
Company is listed;

Meeting is identified by Date and Meeting is board meeting and Company where
        Meeting held on one Date,
        Meeting is board meeting,
        Company held Meeting,
        Meeting is held by one Company;

Person is identified by given-Name and family-Name where
        Person has one given-Name,
        given-Name is of Person,
        family-Name is of Person,
        Person is called at most one family-Name;
Person was born on at most one birth-Date;
Attendance is where
        Person (as Attendee) attended Meeting,
        Meeting was attended by Attendee;
Directorship is where
        Person (as Director) directs Company,
        Company is directed by at least one Director;
Directorship began on one appointment-Date;

Employee is a kind of Person identified by its Nr;
Employee works at one Company,
        Company employs Employee;

Manager is a kind of Employee;
Employee is supervised by at most one Manager [acyclic],
        Manager supervises Employee;
Manager is ceo;
```

Mapping From Facts to ER

...Halpin

- Identifiers define keys
- Decide **absorption** using uniqueness
- **Composition** - makes composite fact types
- Other constraints can be enforced too

SQL Composition

```
CREATE TABLE Attendance (
        AttendeeFamilyName      varchar(48) NULL,
        AttendeeGivenName       varchar(48) NOT NULL,
        MeetingCompanyName      varchar(48) NOT NULL,
        MeetingDate     datetime NOT NULL,
        MeetingIsBoardMeeting   bit NOT NULL,
        UNIQUE(AttendeeGivenName, AttendeeFamilyName, MeetingDate, MeetingIsBoardMeeting, MeetingCompanyName)
)
GO

CREATE TABLE Company (
        CompanyName     varchar(48) NOT NULL,
        IsListed        bit NOT NULL,
        UNIQUE(CompanyName)
)
GO

CREATE TABLE Directorship (
        CompanyName     varchar(48) NOT NULL,
        DirectorFamilyName      varchar(48) NULL,
        DirectorGivenName       varchar(48) NOT NULL,
        AppointmentDate datetime NOT NULL,
        UNIQUE(DirectorGivenName, DirectorFamilyName, CompanyName),
        FOREIGN KEY(CompanyName)
        REFERENCES Company(CompanyName)
)
GO

CREATE TABLE Person (
        FamilyName      varchar(48) NULL,
        GivenName       varchar(48) NOT NULL,
        BirthDate       datetime NULL,
        EmployeeCompanyName     varchar(48) NULL,
        EmployeeManagerNr       int NULL,
        EmployeeNr      int NULL,
        ManagerIsCeo    bit NULL,
        UNIQUE(GivenName, FamilyName)
)
```

Generated SQL - today

- Tables
- Columns
- Primary keys / Unique constraints
- Foreign Key constraints

Generated SQL - coming

- CHECK constraints
- Indexes over views where needed
- Triggers if nec.
- Automatic Migration
  - even delayed migration

Generated Ruby

```ruby
require 'activefacts/api'

module CompanyDirector

  class CompanyName < String
    value_type :length => 48
  end

  class EmployeeNr < SignedInteger
    value_type :length => 32
  end

  class Name < String
    value_type :length => 48
  end

  class Company
    identified_by :company_name
    one_to_one :company_name                    # See CompanyName.company
    maybe :is_listed
  end

  class Meeting
    identified_by :date, :is_board_meeting, :company
    has_one :company                            # See Company.all_meeting
    has_one :date                               # See Date.all_meeting
    maybe :is_board_meeting
  end
```

```
  class Person
    identified_by :given_name, :family_name
    has_one :birth_date, Date                    # See Date.all_person_by_birth_date
    has_one :family_name, Name                   # See Name.all_person_by_family_name
    has_one :given_name, Name                    # See Name.all_person_by_given_name
  end

  class Attendance
    identified_by :attendee, :meeting
    has_one :attendee, Person                    # See Person.all_attendance_by_attendee
    has_one :meeting                             # See Meeting.all_attendance
  end

  class Directorship
    identified_by :director, :company
    has_one :company                             # See Company.all_directorship
    has_one :director, Person                    # See Person.all_directorship_by_director
    has_one :appointment_date, Date              # See Date.all_directorship_by_appointment_date
  end

  class Employee < Person
    identified_by :employee_nr
    has_one :company                             # See Company.all_employee
    one_to_one :employee_nr                      # See EmployeeNr.employee
    has_one :manager                             # See Manager.all_employee
  end

  class Manager < Employee
    maybe :is_ceo
  end

end
```

Ruby API

- Uses meta programming
- Relationships create both methods
- Implements multiple inheritance
- Excludes Readings (for now)

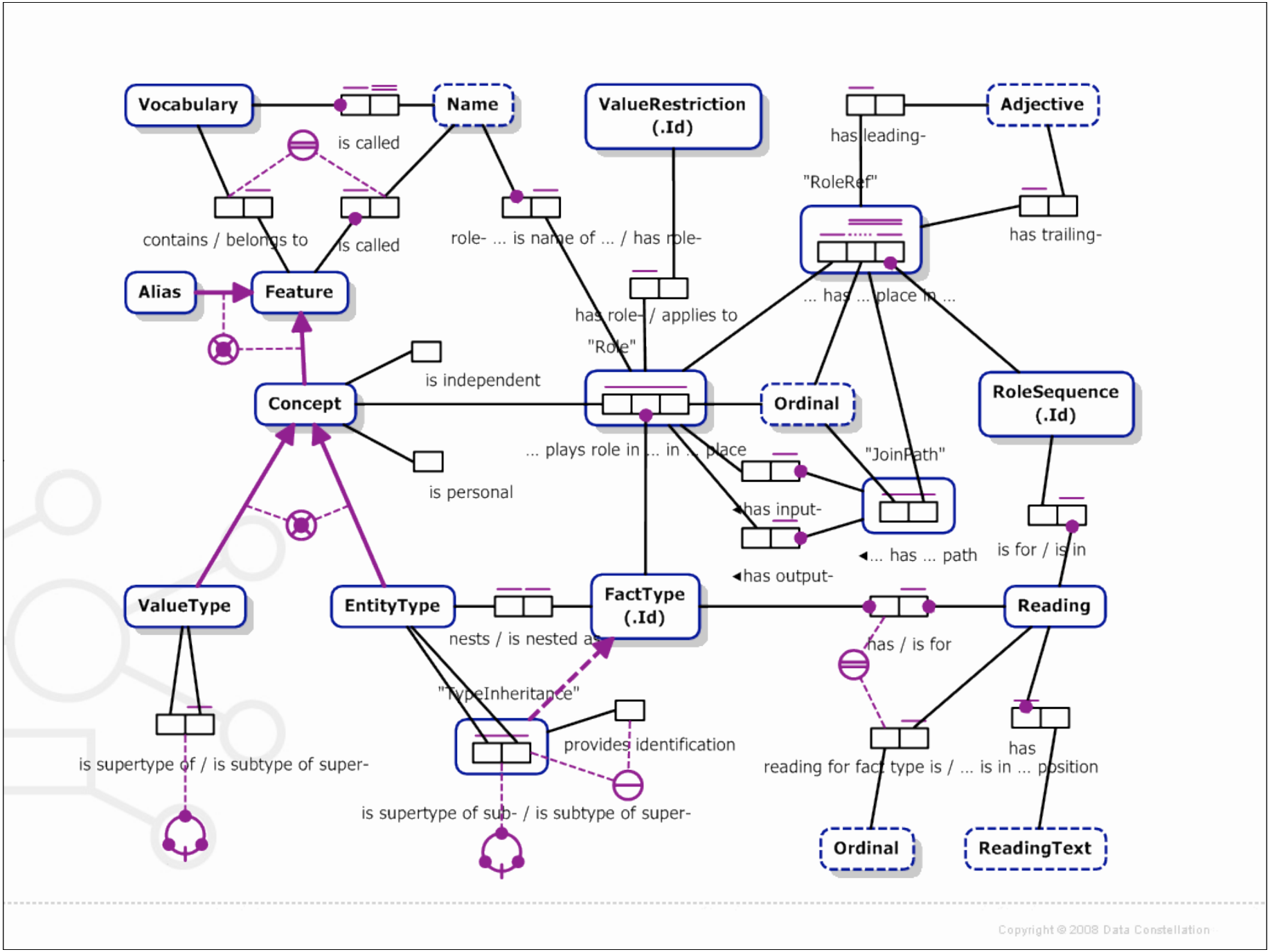
No need to generate the code

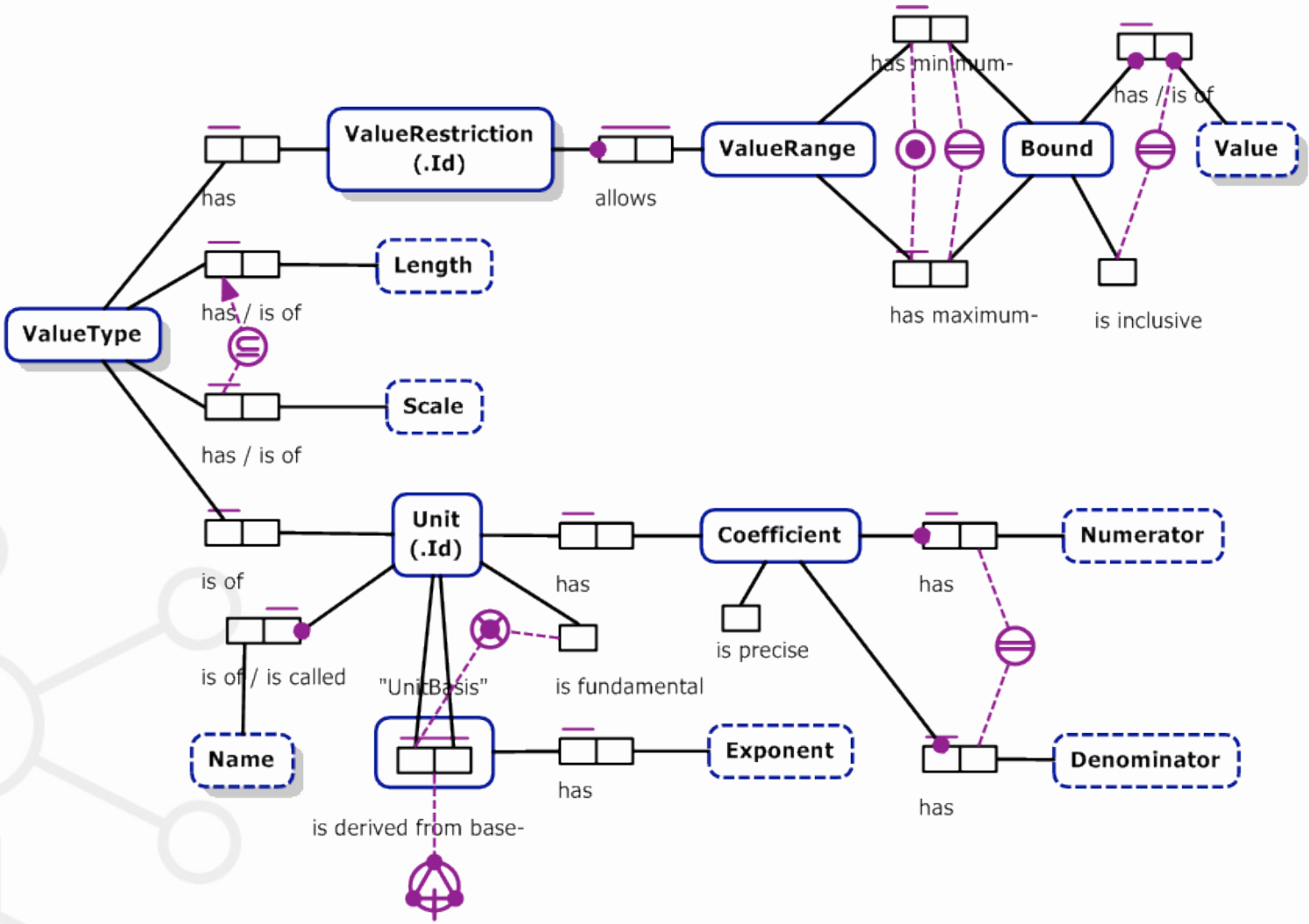
- Simply ‘require’ the CQL
- ActiveFacts generates and eval’s Ruby
- Open classes allow you to extend them
- No need to edit generated code

API Documentation

- Every Role has a method
- ORM2 diagram **is** API documentation
- Could also generate documentation

<DogFood>


Vocabulary
Name
is called
contains / belongs to
is called
Alias
Feature
ValueRestriction
(.Id)
role- ... is name of ... / has role-
has role- / applies to
"Role"
Adjective
has leading-
"RoleRef"
has trailing-
... has ... place in ...
Concept
is independent
is personal
... plays role in ... in ... place
Ordinal
RoleSequence
(.Id)
"JoinPath"
has input-
has output-
... has ... path
is for / is in
ValueType
EntityType
nests / is nested as
FactType
(.Id)
Reading
has / is for
"TypeInheritance"
provides identification
is supertype of / is subtype of super-
is supertype of sub- / is subtype of super-
reading for fact type is / ... is in ... position
has
Ordinal
ReadingText
Copyright © 2008 Data Constellation



ActiveFacts uses its generated Ruby metamodel code

Programming and Persistence

Constellation API

- Constellation is a population of fact instances over a vocabulary

Constellation API

- A Vocabulary is a Ruby module
- Every concept is a class
- Every instance is of a concept (no raw values)
- No duplicate instances
- Fully cross-referenced
- No new() - just assert / deny

Persistence (not implemented yet)

- One query for each user action
- Each query yields a constellation
- Assert/deny any facts, then
- One save()
- Constraint enforcement
- Changes get saved transactionally

Queries

(work in progress)

Simple Query

Person has given-Name ‘Daniel’?

Deriving Fact Types

family-Name controls Company **:**
  Person directs Company,
  Person has family Name;

family Name controls Company ‘Acme, Inc’?
family Name controls Company?
family Name ‘Heath’ controls Company?

Query nesting

- A query can invoke derived fact types
- Sub-queries return distinct instances
  (handles SQL's DISTINCT and GROUP BY)
- Aggregation functions as per SQL
- Goal: more power than SQL SELECT

“returning”

family Name controls Company :
  Person directs Company,
  Person has family Name,
  returning Person,
    by Company;

- “returning” makes Person.given_name available
- “returning by” applies sorting
- Inner join semantics apply if family Name is unknown

“returning” is transitive

normal stuff for Person:
maybe Person was born on birth Date,
maybe Person is Employee,
Employee has EmployeeNr,
returning birth Date, EmployeeNr;

Person is called family-Name ‘Smith’,
normal stuff for Person?

Transitive queries

Employee works under Manager :
  Employee is supervised by Manager [transitive];

Employee works under Manager 473?

either/or

family Name is associated with Company **:**
Person (as Director) directs Company or
Person works at Company,
Person has family Name;

Units

Pane has Area:
  Pane of glass has Width,
  Pane of glass has Height,
  Width * Height = Area;

large Pane:
  Pane has Area, Area >= 5 foot^2;

large Pane?

Width and Height may be stored in millimetres!

Date and Time

```
Person is adult:
   Person was born on birth Date,
   birth Date < Now - 18 years,
      returning birth Date;
```

future: “Person was born on birth Date before 18 years ago”

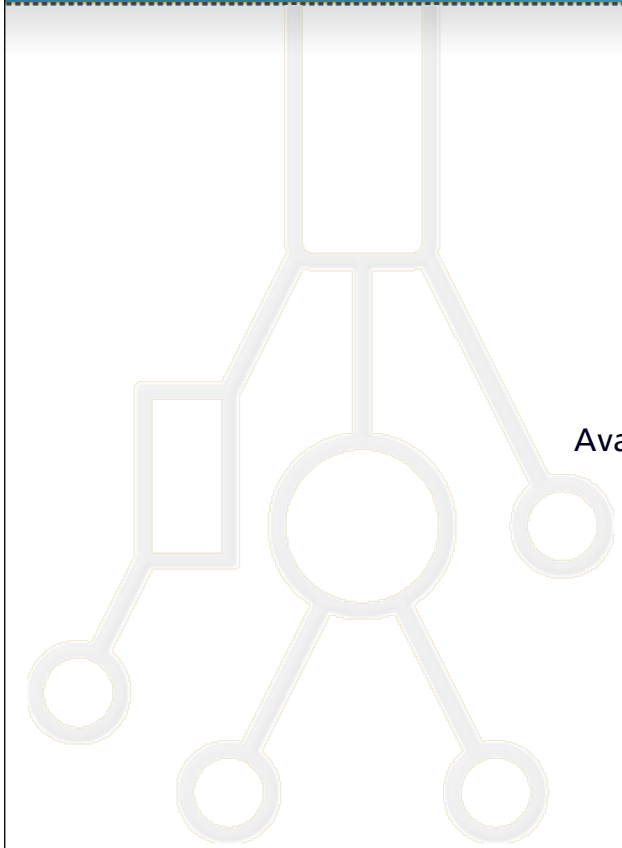
Future work

- API Persistence and DBMS adapters
- Queries, including drag-drop GUI
- Units support
- Other languages (natural and computer)
- APRIMO, a web-hosted semantic designer
- Reverse engineering
- Automatic migration

Demonstrations and Questions

Data Constellation

Agile Information Management and Design



Clifford Heath

Available for consulting and training

http://dataconstellation.com/