

CCL: A Lightweight ORM Embedding in Clean

Bas Lijnse

Patrick van Bommel

Rinus Plasmeijer



Radboud Universiteit Nijmegen



A little about me

Bas Lijnse

PhD Student (final year)

Radboud Universiteit Nijmegen



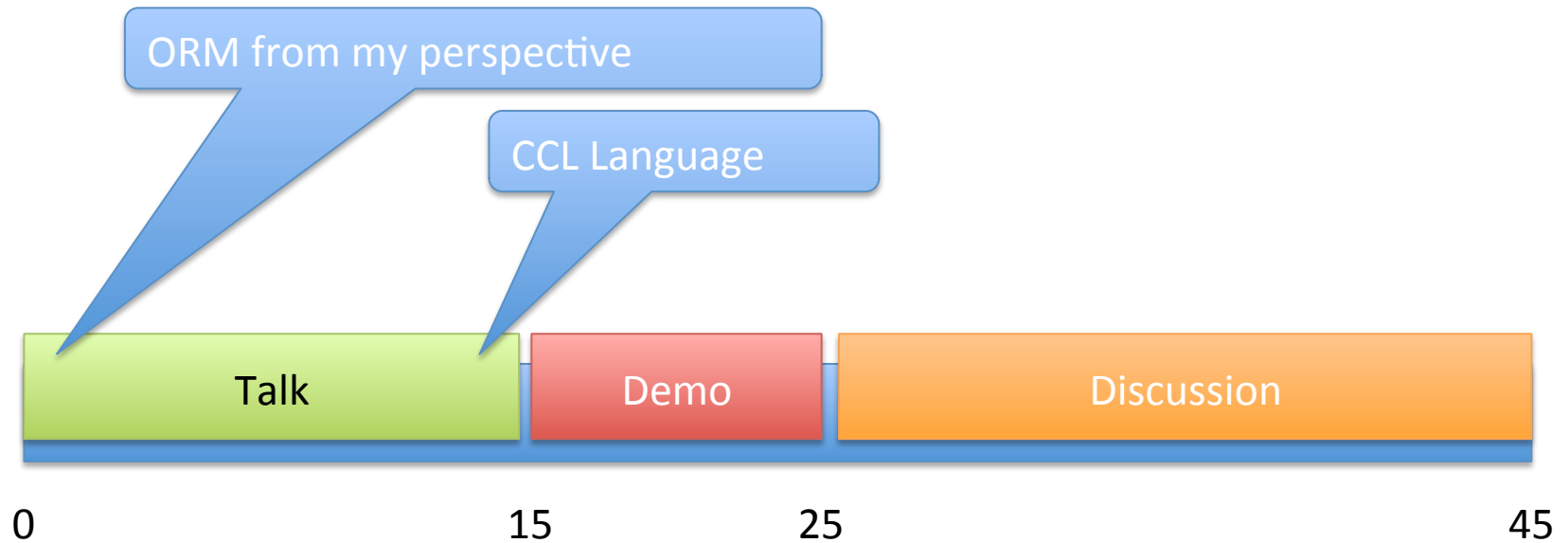
Radboud University
Nijmegen, The Netherlands

Functional Programming Group



Netherlands Defense Academy
Den Helder, The Netherlands
SEWACO/C4I Group

Plan for Today



My Day Job

Generally

Software Development Tools

- Methods, Languages, Libraries, Frameworks

Application Domains

- Crisis Management, Command and Control

Specifically

Functional Programming with Clean

Task-Oriented Programming with iTasks

Functional Programming **with** **Clean**



- Pure lazy functional language
- Developed at Radboud University (since 1984)
- Similar to Haskell (and caml,lisp,scheme,f# etc)
- General purpose language
 - Based on Lambda Calculus with graph reduction semantics
 - Statically Typed (with Hybrid Dynamic types)
 - Higher order functions
 - Algebraic Data Types (Burstall style)

Task-Oriented Programming **with** **iTasks**

Task-Oriented Programming

- Task is primary building block
- High-level basic tasks + composition operators
- Define process and data together
- Specify complete executable systems

A Task is a specified piece of work aiming to produce a result of known **type**.

When **executed** tasks produce (temporary) results which are **observed** in a **controlled** way.



itasks

dynamic workflow system

- Implementation of TOP
- Toolkit for building (prototypes of) TOP applications
- Buzzwords
 - Declarative, Functional Programming in Clean
 - Domain specific language / library in Clean
 - Code generation, Generic Programming

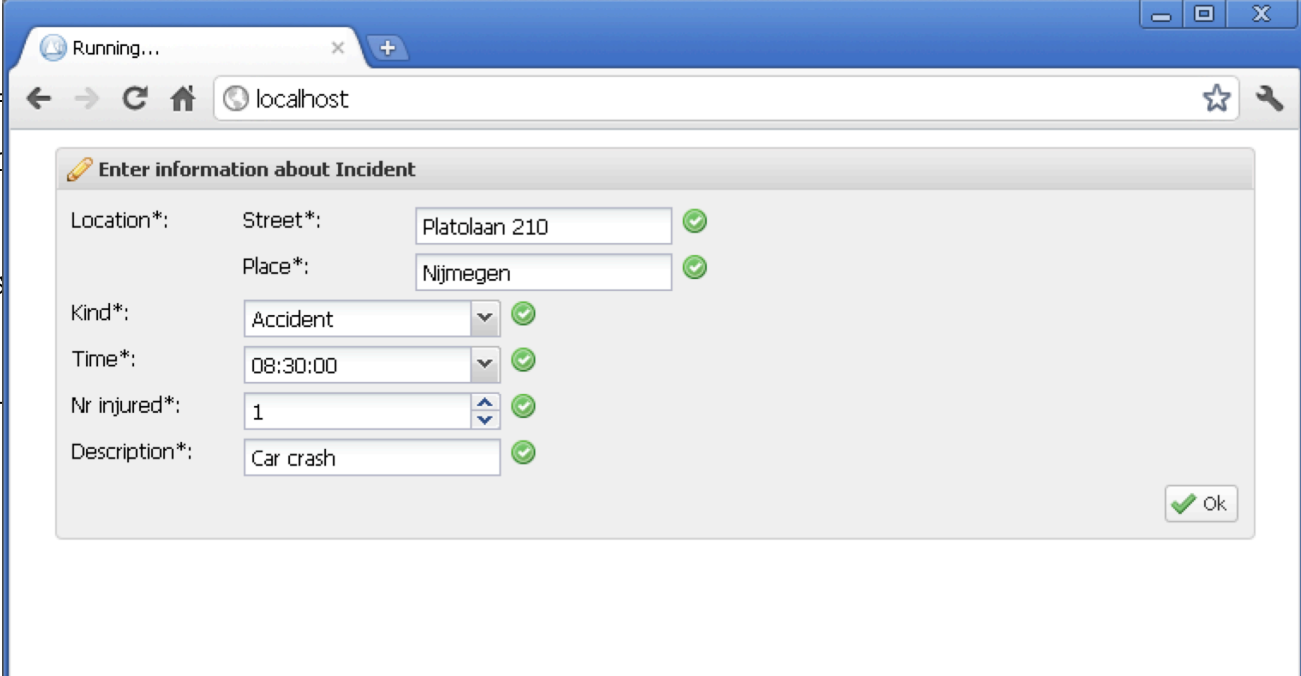
Example Basic Task

```
incidentvb :: Task Incident
incidentvb = enterInformation "Enter...Incident" []

::Incident = {location      :: Location
              ,kind         :: IncidentKind
              ,time         :: Time
              ,nrInjured    :: Int
              ,description   :: String
              }

::IncidentKind =
::Location = {str

derive class iTas
```



The screenshot shows a web browser window with a single tab titled "Running...". The address bar shows "localhost". The main content area displays a form titled "Enter information about Incident". The form contains several input fields, each with a green checkmark indicating it is valid:

- Location*: Street*: Platolaan 210
- Place*: Nijmegen
- Kind*: Accident
- Time*: 08:30:00
- Nr injured*: 1
- Description*: Car crash

An "Ok" button with a green checkmark is located at the bottom right of the form.

Use of TOP & iTasks

- Rapid Prototyping
- “Agile” development
- Task Analysis

The screenshot displays the INCIDONE web application, powered by iTasks, running in a browser window. The interface is for managing incidents, specifically for a kitesurfer injury at Scheveningen beach. The main form is titled 'Coordinate incident' and includes fields for 'Title' and 'Summary'. The 'Title' field contains 'Kitesurfer injured Scheveningen' and the 'Summary' field contains 'Report of kitesurfer with possible injuries in water at Scheveningen beach'. Below the form, there is a table for 'Refinements' and a section for 'Suggestions'.

Title	Description	Completed
Verifiëren melding	De binnengekomen m...	False
Contact maken (lucht)...	pogingen in het werk ...	False

The 'Suggestions' section includes checkboxes for 'Uitkijk spoedbericht (PAN) uitzenden' and 'Voorwaarschuwing'. At the bottom of the interface, there are buttons for 'Add predefined', 'Add custom', 'Add all', 'Add selected', and 'Refresh'.

Types, Types and more Types

```
:: SongSummary =  
{ song_id :: Int  
, title   :: String  
, appears_on [(TrackNo, AlbumName)]  
}  
  
viewSongSummary :: SongId -> Task SongSummary  
viewSongSummary id  
= loadSongSummary id  
>>= viewInformation "Song info:" []
```

```
:: Album =  
{ album_id :: Int  
, title   :: String  
, songs   :: [(TrackNo, SongName)]  
}
```

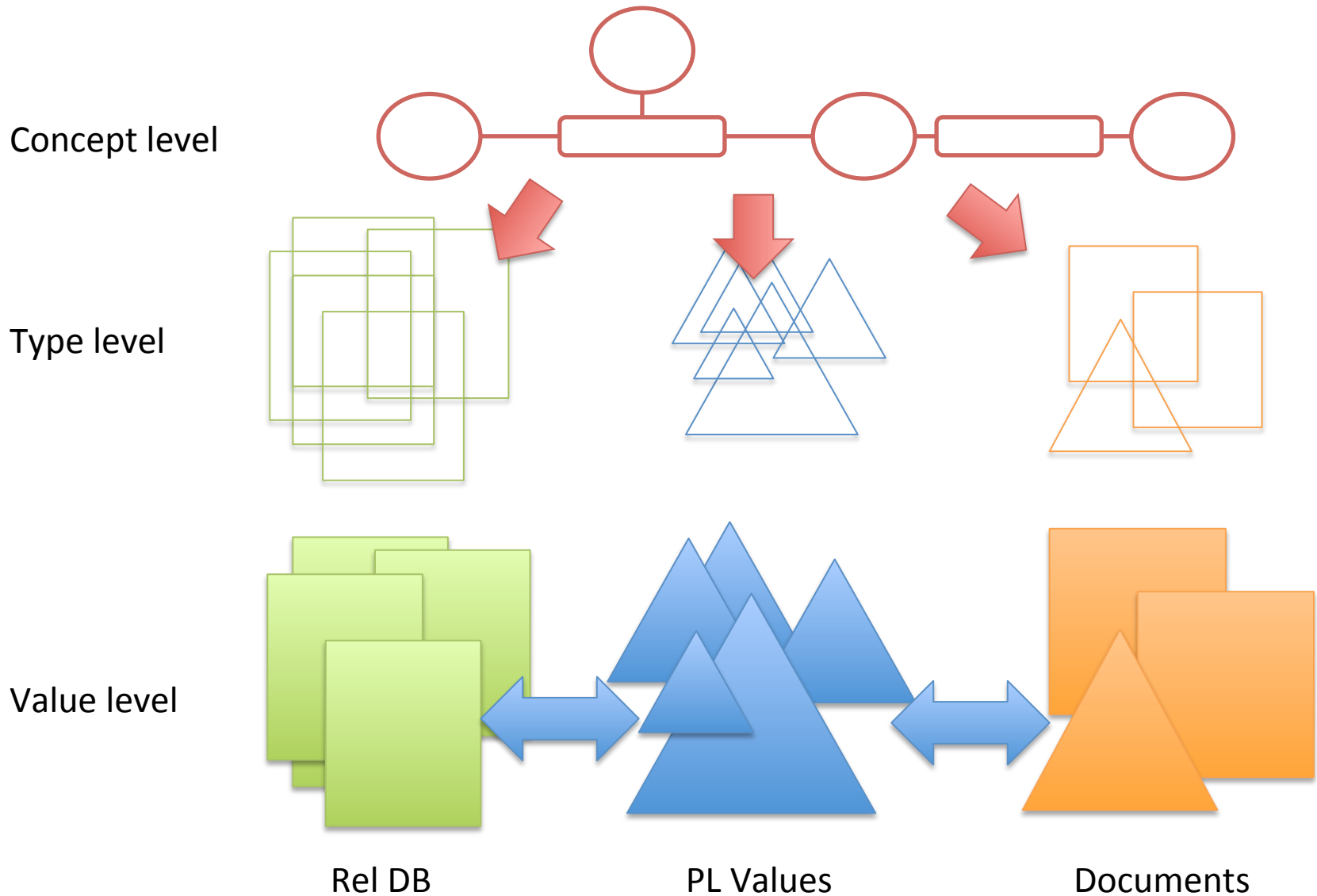
Inverse information

```
:: NewSong =  
{ title      :: String  
, artist     :: String  
, appears_on :: [(TrackNo, AlbumName)]  
}  
  
addSong :: Task NewSong  
addSong  
= enterInformation "New song.." []  
>>= storeSong
```

Same information

Slight variations needed for different tasks
Conceptually related, but compiler can not know

ORM Above All



ORM modeling **with** **CCL**

Concepts in *C*lean

- Textual ORM language
- Extension to Clean
- Provides extra abstraction over Clean types
- Lightweight ORM subset

Why CCL?

- More concise specification of ***conceptually related*** Clean types
- Make conceptual relations explicit
 - Enable more code generation
 - Enable more system visualization
- Drop in language extension (no extra tools)
- Mix conceptual specification with task specification

concept module Music

\$\$ Album

\$\$ AudioBook [A

\$\$ Author

\$\$ MusicAlbum [

\$\$ Artist

\$\$ Song

\$\$ Name = String

\$\$ SongId = Int

\$\$ SongTitle = String

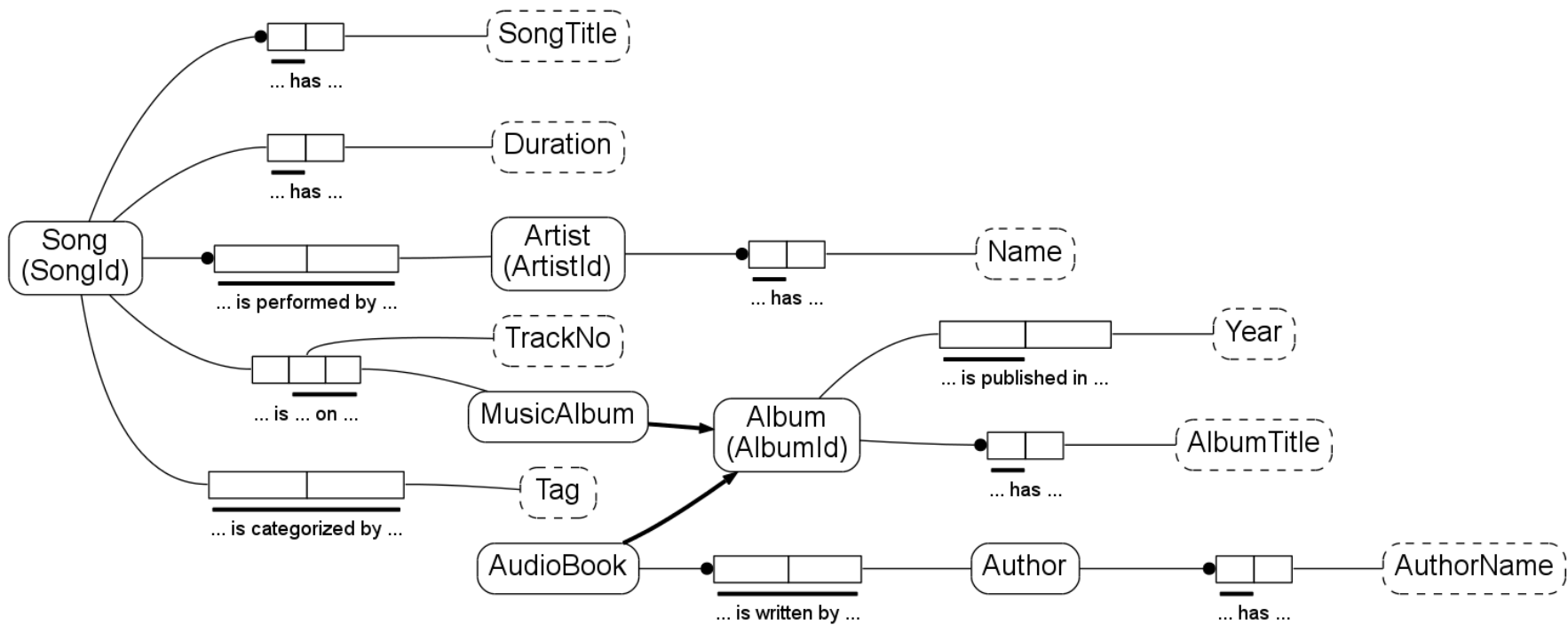
\$\$ AlbumId = Int

\$\$ AlbumTitle = String

\$\$ ArtistId = Int

```
## album_id      = << !!Album >> has << AlbumId >>
## album_title   = << !Album >> has AlbumTitle
## album_year    = << Album >> is published in Year
## song_id       = << !!Song >> has << SongId >>
## title         = << !Song has >> SongTitle
## duration      = << Song >> has Duration
## songs         = Song is << TrackNo on MusicAlbum >>
## performed_by  = << !Song is performed by Artist >>
## tags          = << Song is categorized by Tag >>
## artist_id     = << !!Artist >> has << ArtistId >>
## artist_name   = << !Artist >> has Name
## author_name   = << !Author >> has AuthorName
## author        = << !AudioBook is written by Author >>
```

ring



```
:: Song =  
{song_id :: SongId  
,title :: SongTitle  
,duration :: Maybe Duration  
,songs :: [(TrackNo,String)]  
,performed_by :: [ArtistId]  
,tags :: [Tag]  
}  
  
:: Artist =  
{performed_by :: [SongId]  
,artist_id :: ArtistId  
,artist_name :: Name  
}  
  
:: Album =  
{album_id :: AlbumId  
,album_title :: AlbumTitle  
,album_year :: Maybe Year  
}
```

CCL Language Constructs

Fact Types

Entity Types

Value Types

Sub Types

Uniqueness Constraints

Total Roles

Primary Roles

Fact Container Types

CCL Tools

