

Context Over Corpus: AI Autocompletion for Low-Resource DSLs

India Nunes

XLING

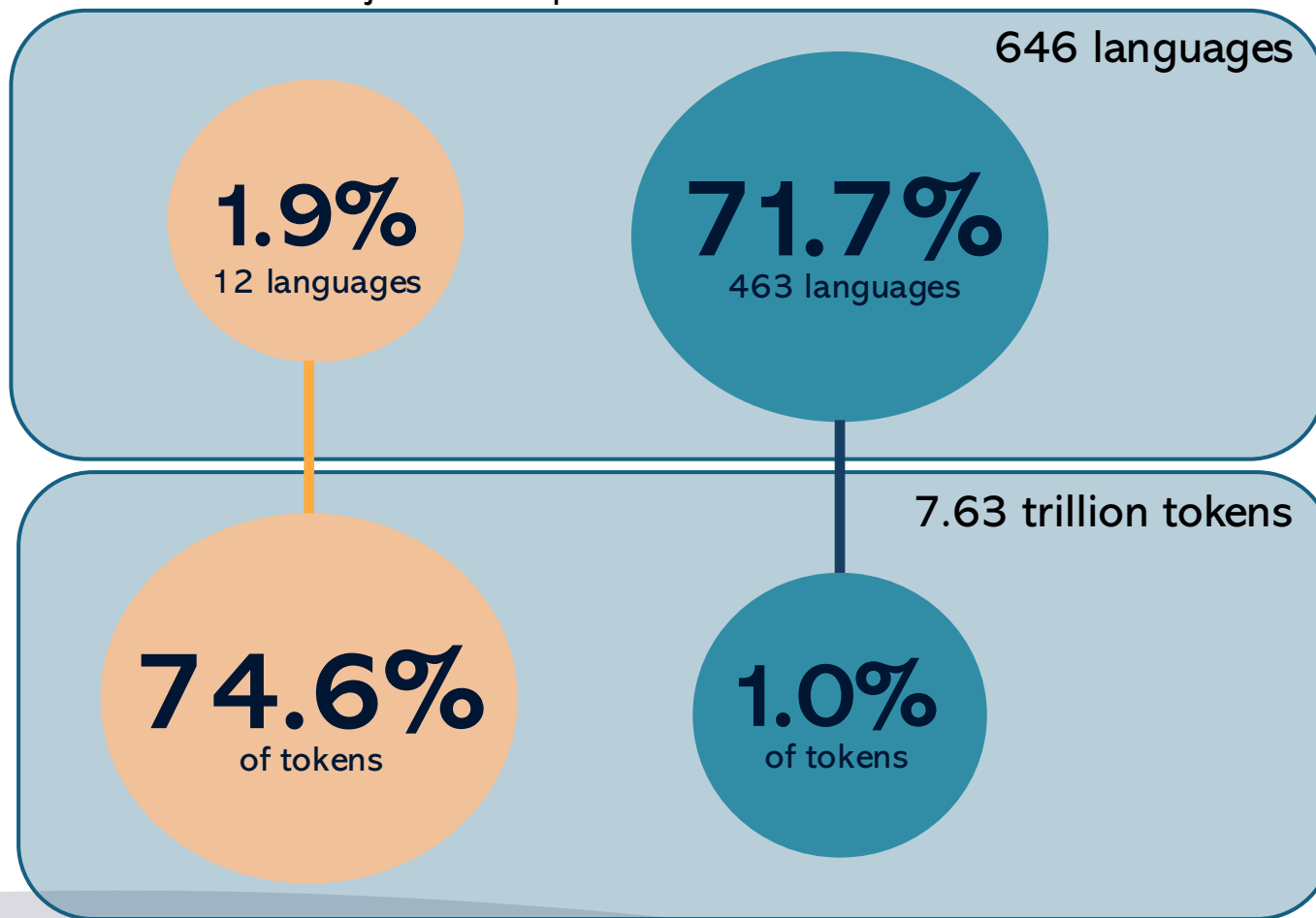
 **LangDev
CON2026**

Torremolinos, 8-9 October, 2026

<https://langdevcon.org>

Low-Resource DSLs

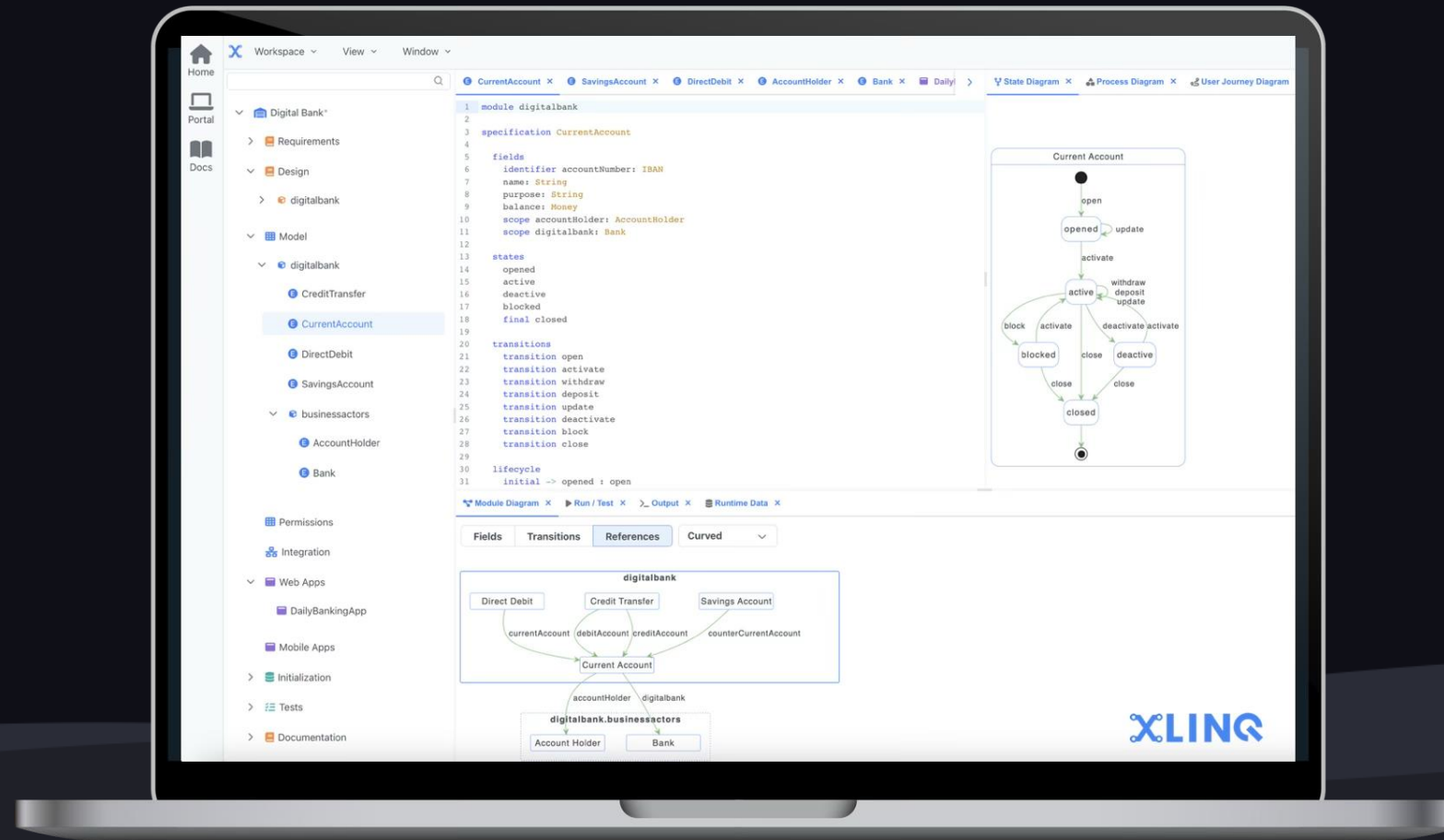
Research across 7 major code corpora [1]



The problem

AI autocompletion for DSLs

XLINQ's Design Center



The goal

Meaningful inline autocompletion for DSLs

```
© PaymentProduct X
1 module referencedata
2
3 specification PaymentProduct
4
5
6 fields
7   identifier id      : String autogen "PPT", "1001"
8   name              : String // Possible values are Normal, Urgent, Cheap
9   description       : String
10  applicableCurrencies : Set[Currency]
11  scope bank         : Bank
12  scope serviceProvider : ServiceProvider
13
14 states
15   active
16   final closed
17
18 transitions
19   transition create
20     parameters
21       set name
22       set description
23       set applicableCurrencies
24       set bank
25     preconditions
26       expect bank is active
27     postconditions
28       set serviceProvider = bank.serviceProvider
29
30   transition close
31
32 lifecycle
33   initial -> active : create
34   active  -> closed : close
35
```

The goal

Meaningful inline autocompletion for DSLs

1. fields

2. states

3. transitions

4. lifecycle

The goal

Meaningful inline autocompletion for DSLs

31		
32	lifecycle	



The goal

Meaningful inline autocompletion for DSLs

```
31  
32 lifecycle
```



lifecycle

```
on create  
  set state to created  
  
on delete  
  set state to deleted
```



lifecycle|

```
on create  
  set state = active  
  
on close  
  set state = closed
```

lifecycle

```
initial -> active : create  
active -> closed : close
```

The goal

Meaningful inline autocompletion for DSLs

```
31  
32 lifecycle  
33   initial -> active : create  
34   active  -> closed : close  
35
```



lifecycle

```
on create  
  set state to created  
  
on delete  
  set state to deleted
```



lifecycle|

```
on create  
  set state = active  
  
on close  
  set state = closed
```

lifecycle

```
initial -> active : create  
active -> closed : close
```

The challenge

HOW?

Maximum capability

Minimum latency



Ground-truth-driven experiments

Systematically test a stochastic output

Use DSL files with **redacted target snippets**

- └ prefix
- └ cursor
- └ suffix

what the user has written

where completion is requested

remainder of the file, when applicable



fields' section body completion

```
1 module accountmanagement
2
3 specification AccountHolder of kind businessactor from template BusinessActor
4 fields
5   identifier id          : String autogen "AH0", "1001"
6   name                   : String
7   involvedParty          : InvolvedParty @SubsetOf(call individualsOnly())
8   individual             : Individual
9   domesticCurrency       : Currency
10  scope accountHolder    : AccountHolder
11  scope bank             : Bank
12  specName               : String
```



fields' section body completion

```
1 module accountmanagement
2
3 specification AccountHolder of kind businessactor from template BusinessActor
4 fields
5   identifier id          : String autogen "AH0", "1001"
6   name                   : |
7
8
9
10
11
12
```



fields' section body completion

```
1 module accountmanagement
2
3 specification AccountHolder of kind businessactor from template BusinessActor
4 fields
5   identifier id          : String autogen "AH0", "1001"
6   name                   : String
7   scope bank             : Bank
8   scope serviceProvider : ServiceProvider
9   optional email         : Email
10  optional phoneNumber    : PhoneNumber
11
12
```



fields' section body completion

```
6   name                : String
7   involvedParty       : InvolvedParty @SubsetOf(call individualsOnly())
8   individual          : Individual
9   domesticCurrency    : Currency
10  scope accountHolder : AccountHolder
11  scope bank           : Bank
12  specName            : String
```

```
name                : String
scope bank           : Bank
scope serviceProvider : ServiceProvider
optional email       : Email
optional phoneNumber : PhoneNumber
```



fields' section body completion

```
fields
```

```
  identifier id : String autogen "AH0", "1001"
```

```
  name : String
```

```
- involvedParty : InvolvedParty @SubsetOf(call individualsOnly())
```

```
- individual : Individual
```

```
- domesticCurrency : Currency
```

```
- scope accountHolder : AccountHolder
```

```
  scope bank : Bank
```

```
- specName : String
```

```
+ scope serviceProvider : ServiceProvider
```

```
+ optional email : Email
```

```
+ optional phoneNumber : PhoneNumber
```

```
Uni Diff Summary
```

```
- ground truth contains field(s):
```

```
  ['id', 'name', 'involvedParty', 'individual',  
   'domesticCurrency', 'accountHolder', 'bank', 'specName']
```

```
- generated field(s):
```

```
  ['id', 'name', 'bank', 'serviceProvider']
```

```
- skipped field(s):
```

```
  ['involvedParty', 'individual', 'domesticCurrency',  
   'accountHolder', 'specName']
```

```
- generated (not in ground truth):
```

```
  ['serviceProvider']
```



Context over corpus

* with diminishing returns!

❑ Across different cursor positions + sections + files:

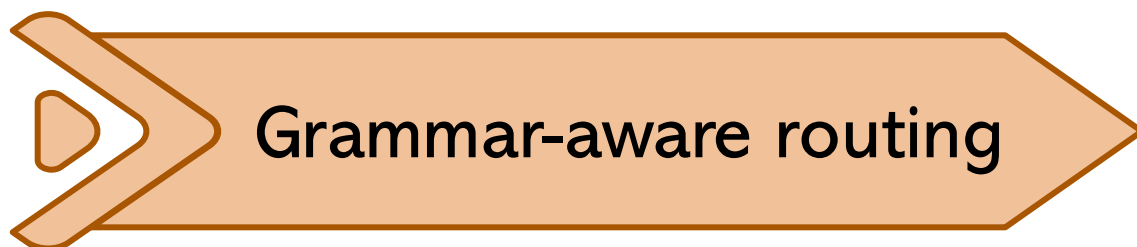
- ❑ instruction/prompt phrasing
- ❑ context scope
- ❑ template granularity
- ❑ documentation depth
- ❑ interaction structure

❑ Techniques applied:

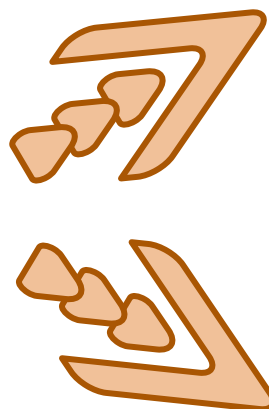
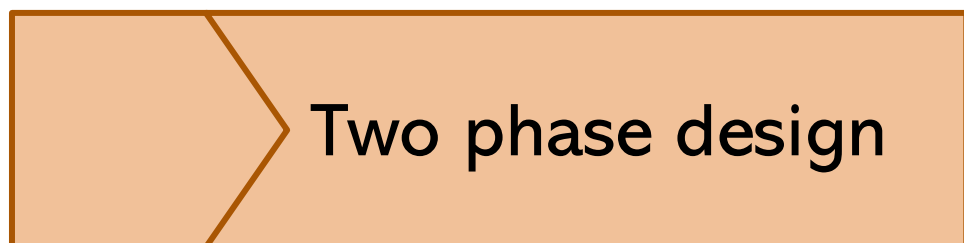
- ❑ in-context learning (ICL)
- ❑ inference-time language acquisition
- ❑ structured multi-turn prompting
- ❑ task decomposition
- ❑ few-shot ICL
- ❑ prompt & context engineering
- ❑ implicit chain-of-thought (CoT)
- ❑ instruction + prompt separation



Resulting architecture



1. fields
2. states
3. transitions
4. lifecycle



Phase 1

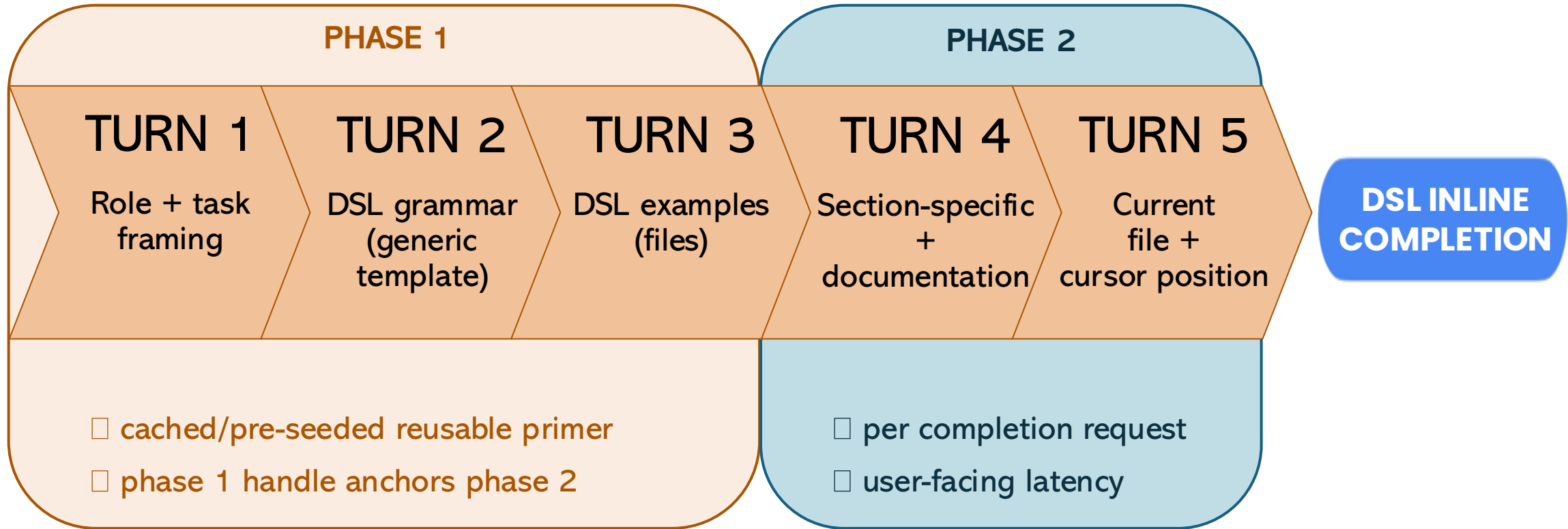
primer agnostic to:

- section (i.e., cursor position)
- current file

Phase 2

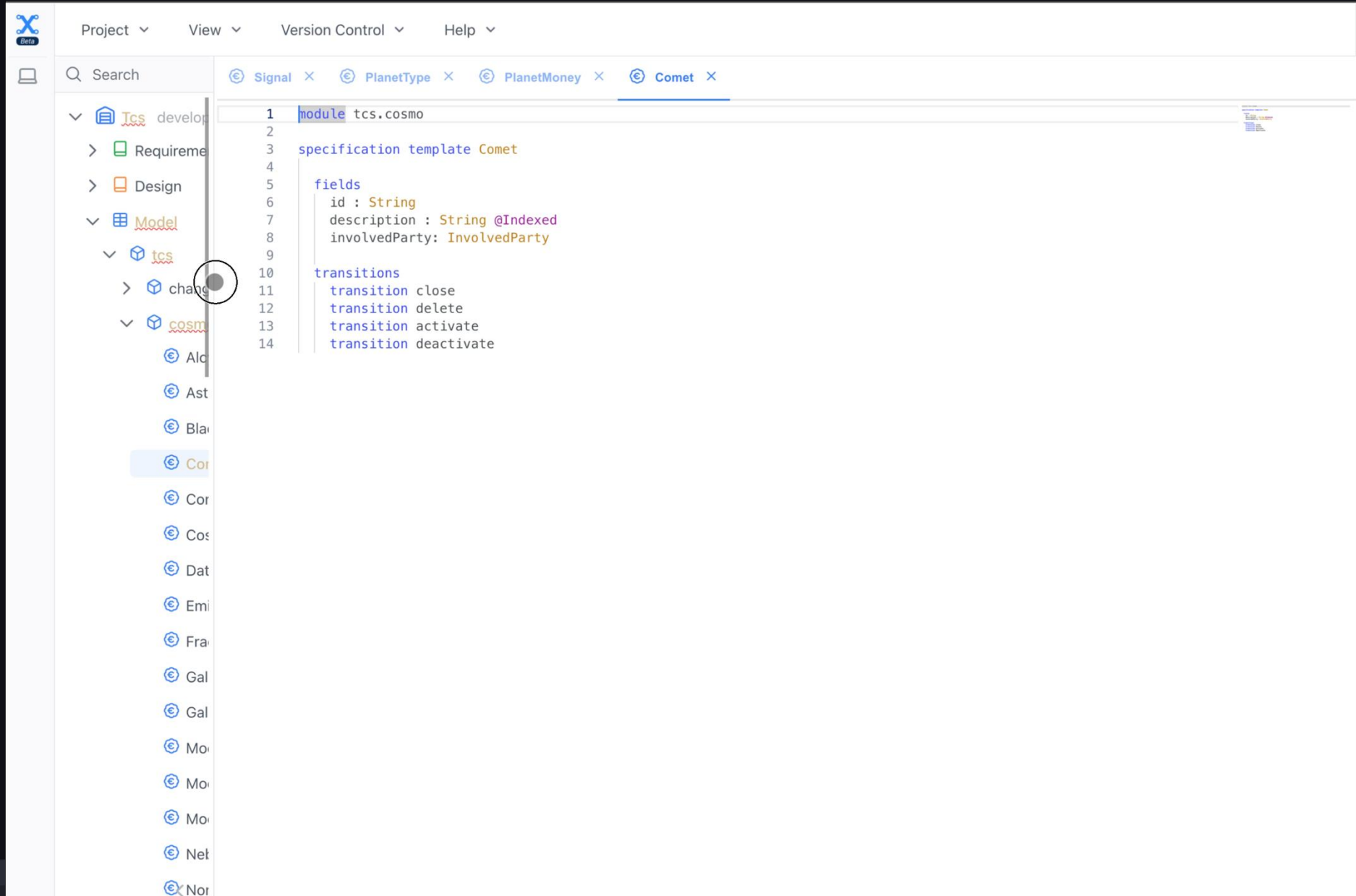
- current cursor position and file
- section-specific context
- completion generation

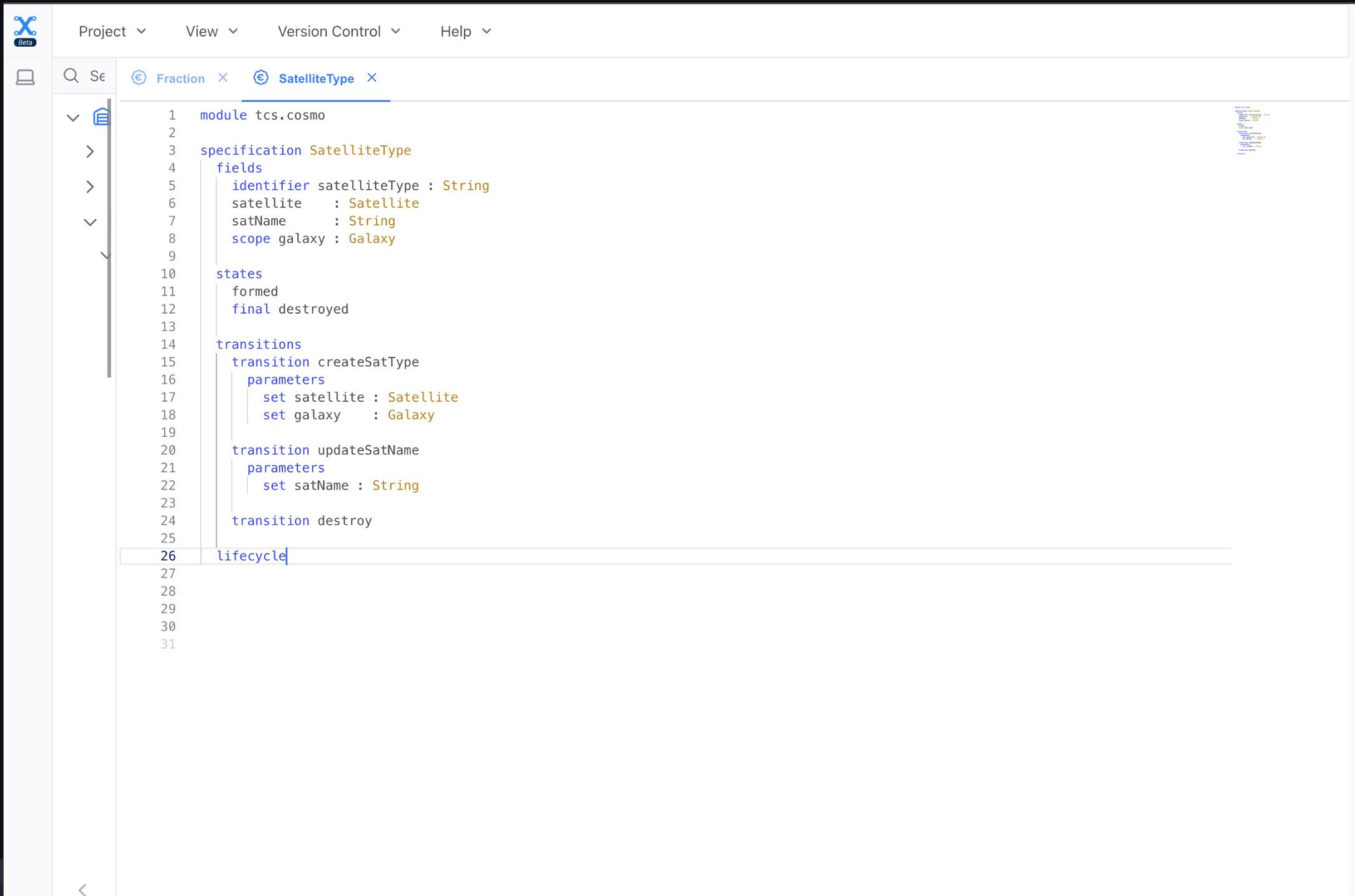
Resulting architecture



Demo







Fraction ×

```
1 module tcs.cosmo
2
3 specification Fraction
4   fields
5     identifier id : String
6     numerator : Integer
7     denominator : Integer
8     scope universe : Universe
9
10  states
11    active
12    final finalised
13
14  transitions
15    transition create
16      parameters
17        set numerator : Integer
18        set denominator : Integer
19        set universe : Universe
20
21
22    transition update
23      parameters
24        set numerator : Integer
25        set denominator : Integer
26
27
28    transition finalize
29
30  lifecycle
31    initial -> active : create
32    active -> active : update
33    active -> finalised : finalize
34
35
36
```

View ▾ Version Control ▾ Help ▾

Fraction ×

```
1 module tcs.cosmo
2
3 specification Fraction
4   fields
5     identifier id : String
6     numerator : Integer
7     denominator : Integer
8     approximation : Real
9     scope universe : Universe
10
11   states
12     active
13     final finalised
14
15   transitions
16     transition create
17       parameters
18         set numerator : Integer
19         set denominator : Integer
20         set universe : Universe
21
22
23     transition update
24       parameters
25         set numerator : Integer
26         set denominator : Integer
27
28     transition
29
30
31     transition finalize
32
33   lifecycle
34     initial -> active : create
35     active -> active : update
36     active -> active : approximate
37     active -> finalised : finalize
38
39
40
```

transition simplify
parameters
 set approximation : Real

preconditions
 expect denominator != 0
 | fail with error FRAC0001

postconditions
 set approximation = numerator / denominator

What is achieved

inference-time autocompletion for a low-resource DSL

What remains open

generalisability - what does it take to do this for another DSL (yours?)

And a lot more... ask me in the break!



Questions?

Please find me afterwards.

Thank you!



india@xling.io



Build next generation software with
Business Knowledge as Code

XLING

Delivering.
Reliable.
Software.
Faster.

