APIphany: Type-Directed Program Synthesis For REST APIs

Zheng Guo*, David Cao*, Davin Tjong*, Jean Yang†, Cole Schlesinger‡, Nadia Polikarpova*

* University of California San Diego
† Akita Software
Slack API: Retrieve all member emails from a slack channel

Given the name of a slack channel, is there a way to retrieve a list of emails of all the members in that channel? I tried looking in the slack api docs but couldn't find the method I need to make this happen (https://api.slack.com/methods).

16

Share  Improve this question  Follow
Overview / Running Example

**Task:** retrieve all member emails from a Slack channel given the channel name

```
channel_name => {
    conversations_list()
}
```
**Task:** retrieve all member emails from a Slack channel given the channel name

```javascript
channel_name => {
    conversations_list()
    .filter(c => c.name == channel_name)
}
```
Task: retrieve all member emails from a Slack channel given the channel name

```javascript
channel_name => {
    conversations_list()
    .filter(c => c.name == channel_name)
    .map(c => {
        conversations_members(c.id)
    })
}
```
**Task:** retrieve all member emails from a Slack channel given the channel name

```
channel_name => {
    conversations_list()
    .filter(c => c.name == channel_name)
    .map(c => {
        conversations_members(c.id)
        .map(uid => {
            let u = users_info(user=uid)
        })
    })
}
```
Task: retrieve all member emails from a Slack channel given the channel name

```javascript
channel_name => {
    conversations_list() // filters conversations based on channel name
    .filter(c => c.name === channel_name)
    .map(c => {
        conversations_members(c.id) // maps members of the conversation
        .map(uid => {
            let u = users_info(user=uid) // gets user information
            return u.profile.email
        })
    })
}
```
Task: retrieve all member emails from a Slack channel given the channel name

```javascript
channel_name => {
  conversations_list()
    .map(uid => {
      let u = users_info(user=uid)
      return u.profile.email
    })
}
```
Overview / Program Synthesis

Specifications
Overview / Program Synthesis

Specifications

Components
Overview / Program Synthesis

Specifications

Components
Overview / Program Synthesis

Specifications

Components

Synthesizer
Overview / Program Synthesis

Specifications

Components

Synthesizer
Overview / **Program Synthesis**

- **Specifications**
- **Components**
- **Synthesizer**
- **Programs**
What Are Good Specifications For REST APIs?
Specifications / Examples as Specifications?

I/O examples
Specifications / **Examples as Specifications?**

**I/O examples**

😊 side effects
I/O examples

- Side effects
- Large objects
Specifications / Examples as Specifications?

I/O examples

- side effects
- large objects

NOT FIT FOR PURPOSE
Specifications / **NL as Specifications?**

<table>
<thead>
<tr>
<th>I/O examples</th>
<th>natural language</th>
</tr>
</thead>
<tbody>
<tr>
<td>🙁 side effects</td>
<td></td>
</tr>
<tr>
<td>🙁 large objects</td>
<td>NOT FIT FOR PURPOSE</td>
</tr>
</tbody>
</table>
Specifications / **NL as Specifications?**

**I/O examples**
- Side effects
- Large objects

**Natural language**
- Too vague

*NOT FIT FOR PURPOSE*
Specifications / **NL as Specifications?**

<table>
<thead>
<tr>
<th>I/O examples</th>
<th>natural language</th>
</tr>
</thead>
<tbody>
<tr>
<td>☹ side effects</td>
<td>☹ too vague</td>
</tr>
<tr>
<td>☹ large objects</td>
<td></td>
</tr>
</tbody>
</table>
Specifications / *Types as Specifications?*

I/O examples
- Side effects
- Large objects

Natural language
- Too vague

Types

*NOT FIT FOR PURPOSE*
Specifications / **Types as Specifications?**

<table>
<thead>
<tr>
<th>I/O examples</th>
<th>natural language</th>
<th>types</th>
</tr>
</thead>
<tbody>
<tr>
<td>☹ side effects</td>
<td>☹ too vague</td>
<td></td>
</tr>
<tr>
<td>☹ large objects</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*NOT FIT FOR PURPOSE*
Specifications / Types as Specifications?

I/O examples

- side effects
- large objects

natural language

- too vague

types

- coarse-grained

\[ \text{conversations\_members} :: \text{String} \rightarrow [\text{String}] \]
Our Contribution / **Semantic Types as Specifications**!

- **I/O examples**
  - 😞 side effects
  - 😞 large objects

- **natural language**
  - 😞 too vague

- **semantic types**
  - 😊 coarse-grained
Our Contribution / APIPhany

APIPhany

A program synthesizer for REST APIs guided by semantic types
APIphony / Architecture

OpenAPI spec

1. semantic type construction

execution traces

type mining

semantic library
[1] Component-based synthesis for complex APIs. Feng et al. POPL'17
type mining

1. semantic type construction
2. semantic type inference

OpenAPI spec

execution traces

OpenAPI spec

execution traces

type query

semantic library

type-directed synthesis

1,2

programs

retrospective execution

ranked programs

[1] Component-based synthesis for complex APIs. Feng et al. POPL’17
APIphany / Architecture

OpenAPI spec

1. type mining
   - semantic type construction
   - semantic type inference

2. semantic type construction
   - semantic library

3. type-directed synthesis
   - programs

4. retrospective execution
   - ranked programs

[1] Component-based synthesis for complex APIs. Feng et al. POPL’17
Task: retrieve all member emails from a Slack channel given the channel name

Type query: ?
Task: retrieve all member emails from a Slack channel given the channel name

Type query: "channel name" → ["user email"]
Task: retrieve all member emails from a Slack channel given the channel name

Type query:  “channel name” → [“user email”]
**Task:** retrieve all member emails from a Slack channel given the channel name

**Type query:** “channel name” → [“user email”]

**Insight 1: object fields as types!**
Objects

User

{  
id :: String  
, profile :: Profile  
}

Profile

{  
email :: String  
, phone :: String  
}

Channel

{  
creator :: String  
, name :: String  
, id :: String  
}
Objects

User { id :: User.id 
, profile :: Profile }

Profile { email :: String 
, phone :: String }

Channel { creator :: String 
, name :: String 
, id :: String }

Type Mining / Semantic Types
Objects

User

{ id :: User.id
   , profile :: Profile }

Profile

{ email :: Profile.email
   , phone :: Profile.phone }

Channel

{ creator :: Channel.creator
   , name :: Channel.name
   , id :: Channel.id }
Task: retrieve all member emails from a Slack channel given the channel name

Type query: Channel.name → [Profile.email]
OpenAPI spec

execution traces

APIphany / Architecture

[1] Component-based synthesis for complex APIs. Feng et al. POPL’17
<table>
<thead>
<tr>
<th>Objects</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>users_info :: String → User</td>
</tr>
<tr>
<td></td>
<td>conversations_members :: String → [String]</td>
</tr>
<tr>
<td>Profile</td>
<td>conversations_list :: [Channel]</td>
</tr>
<tr>
<td>Channel</td>
<td></td>
</tr>
</tbody>
</table>

```
User { id :: User.id,
      profile :: Profile }

Profile { phone :: Profile.phone,
           email :: Profile.email }

Channel { creator :: Channel.creator,
           name :: Channel.name,
           id :: Channel.id }
```
<table>
<thead>
<tr>
<th>Objects</th>
<th>Methods</th>
</tr>
</thead>
</table>
| User  
  { id :: User.id  
    , email :: Profile.email  
  } | conversations_members :: String → [String] |
| Channel  
  { creator :: Channel.creator  
    , name :: Channel.name  
    , id :: Channel.id  
  } | conversations_list :: [Channel]           |

Insight 2: mine from traces!
**Type Mining / Semantic Type Inference**

**Invocation**

```java
users_info("UJ5RHEG4S") ==
{ "id": "UJ5RHEG4S",
  "name": "demo_user",
  "profile":
    { "email": "xyz@gmail.com"
    ...
  }
}
```

**Method Spec Type**

```
users_info :: String → User
```
**Type Mining / Semantic Type Inference**

**Invocation**

```java
users_info("UJ5RHEG4S") ==
{
  "id": "UJ5RHEG4S",
  "name": "demo_user",
  "profile":
    {
      "email": "xyz@gmail.com"
    }
}
```

**Method Spec Type**

```
users_info :: String → User
```
Type Mining / Semantic Type Inference

Invocation

users_info("UJ5RHEG4S") =
{
  "id": "UJ5RHEG4S",
  "name": "demo_user",
  "profile":
  {
    "email": "xyz@gmail.com"
  ...
  }
}

Method Spec Type

users_info :: String → User
Type Mining / **Semantic Type Inference**

**Invocation**

```javascript
users_info("UJ5RHEG4S")
{
  "id": "UJ5RHEG4S",
  "name": "demo_user",
  "profile":
    {
      "email": "xyz@gmail.com"
    }
}
```

**Method Spec Type**

```
users_info :: String -> User
```

User.id
Invocation

```javascript
users_info("UJ5RHEG4S") = {
  "id": "UJ5RHEG4S",
  "name": "demo_user",
  "profile": {
    "email": "xyz@gmail.com"
  }
}
```

Method Spec Type

```
users_info :: User.id → User
```
### Objects

<table>
<thead>
<tr>
<th>Class</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td><code>id</code> :: User.id, <code>profile</code> :: Profile</td>
</tr>
<tr>
<td>Profile</td>
<td><code>phone</code> :: Profile.phone, <code>email</code> :: Profile.email</td>
</tr>
<tr>
<td>Channel</td>
<td><code>creator</code> :: Channel.creator, <code>name</code> :: Channel.name, <code>id</code> :: Channel.id</td>
</tr>
</tbody>
</table>

### Methods

- `users_info` :: **User.id** → User
- `convs_members` :: **Channel.id** → [ **User.id** ]
- `convs_list` :: [Channel]
<table>
<thead>
<tr>
<th>Objects</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>User { id :: User.id, email :: Profile.email }</td>
<td>convs_members :: Channel.id → [ User.id ]</td>
</tr>
<tr>
<td>Channel { creator :: Channel.creator, name :: Channel.name, id :: Channel.id }</td>
<td>convs_list :: [ Channel ]</td>
</tr>
</tbody>
</table>
Type Mining / **Semantic Library**

**Objects**

- **User**
  
  ```
  { id :: User.id 
    , profile :: Profile }
  ```

- **Profile**
  
  ```
  { phone :: Profile.phone 
    , email :: Profile.email }
  ```

- **Channel**
  
  ```
  { creator :: Channel.creator 
    , name :: Channel.name 
    , id :: Channel.id }
  ```

**Methods**

- **users_info** :: User.id → User
- **convs_members** :: Channel.id → [ User.id ]
- **convs_list** :: [Channel]

*A user ID*
Objects

User

{ id :: User.id,
  profile :: Profile }

Profile

{ phone :: Profile.phone,
  email :: Profile.email }

Channel

{ creator :: Channel.creator,
  name :: Channel.name,
  id :: Channel.id }

Methods

users_info :: User.id → User

convs_members :: Channel.id → [ User.id ]

convs_list :: [Channel]

A user ID

Traces!
Type Mining / **Semantic Type Inference**

Invocation

```json
{
  "id": "C123",
  "name": "general",
  "creator": "UJ5RHEG4S",
  ...
}
```

Object Spec Type

```json
Channel {
  creator :: Channel.creator,
  name :: Channel.name,
  id :: Channel.id
}
```
Type Mining / Semantic Type Inference

Invocation

```
{ "id": "C123"
, "name": "general"
, "creator": "UJ5RHEG4S"
, ...
}
```

Object Spec Type

```
Channel { creator :: Channel.creator
, name :: Channel.name
, id :: Channel.id }
```

User.id
Type Mining / Semantic Type Inference

Invocation

```json
{
  "id": "C123",
  "name": "general",
  "creator": "UJ5RHEG4S",
  ...
}
```

Object Spec Type

```json
Channel
{
  creator :: User.id,
  name :: Channel.name,
  id :: Channel.id
}
```
Objects

User
{ id :: User.id
, profile :: Profile }

Profile
{ phone :: Profile.phone
, email :: Profile.email }

Channel
{ creator :: User.id
, name :: Channel.name
, id :: Channel.id }

Methods

users_info :: User.id → User
convs_members :: Channel.id → [ User.id ]
convs_list :: [Channel]
APIphany / Architecture

OpenAPI spec
execution traces

1. type mining
   - semantic type construction
   - semantic type inference

2. semantic library

3. type-directed synthesis
   - type query
   - ranked programs

[1] Component-based synthesis for complex APIs. Feng et al. POPL’17
APIphany / Architecture

- OpenAPI spec
- execution traces

1. type mining
   - semantic type construction
   - semantic type inference

2. semantic type inference

3. type-directed synthesis

- type query

- retrospective execution

- ranked programs
Type-Directed Synthesis / Program Ranking

Channel.Name $\rightarrow$ [Profile.Email]

```
channel_name $\Rightarrow$ {
  conversations_list()
    .filter(c $\Rightarrow$ c.name $\equiv$ channel_name)
    .map(c $\Rightarrow$ { conversations_members(c.id)
        .map(uid $\Rightarrow$ { let u = users_info(user=uid)
            return u.profile.email })}
})
```

Desired Solution
Channel.Name → [Profile.Email]

Desired Solution

```
channel_name => {
  conversations_list()
    .filter(c => c.name == channel_name)
    .map(c => {
      conversations_members(c.id)
        .map(uid => {
          let u = users_info(user=uid)
          return u.profile.email })
    })
}
```

Candidate #1

```
channel_name => {
  conversations_list()
    .filter(c => c.name == channel_name)
    .map(c => {
      let uid = c.creator
      let u = users_info(user=uid)
      return u.profile.email })
}
```
Channel.Name → [Profile.Email]

Desired Solution

```javascript
channel_name => {
  conversations_list()
    .filter(c => c.name === channel_name)
    .map(c => {
        conversations_members(c.id)
          .map(uid => {
            let u = users_info(user=uid)
            return u.profile.email
          })
    })
}
```

```
channel_name => {
  conversations_list()
    .filter(c => c.name === channel_name)
    .map(c => {
        let uid = c.creator
        let u = users_info(user=uid)
        return u.profile.email
    })
}
```

Always returns a singleton array
Channel.Name → [Profile.Email]

Desired Solution

```javascript
channel_name => {
  conversations_list()
    .filter(c => c.name === channel_name)
    .map(c => { conversations_members(c.id)
             .map(uid => { let u = users_info(user=uid)
                              return u.profile.email })
       }))
}
```

Candidate #2

```javascript
channel_name => {
  conversations_open()
    .filter(c => c.name === channel_name)
    .map(c => { conversations_members(c.id)
                       .map(uid => { let u = users_info(user=uid)
                                      return u.profile.email })
       }))
}
```
Type-Directed Synthesis / Program Ranking

Channel.Name → [Profile.Email]

```
channel_name => {
    conversations_list().filter(c => c.name == channel_name).
    .map(c => {
        conversations_members(c.id).
        .map(uid => {
            let u = users_info(user=uid)
            return u.profile.email
        })
    })
}
```

Desired Solution

```
channel_name => {
    conversations_open().filter(c => c.name == channel_name).
    .map(c => {
        conversations_members(c.id).
        .map(uid => {
            let u = users_info(user=uid)
            return u.profile.email
        })
    })
}
```

Always fails
Type-Directed Synthesis / Program Ranking

Candidate #2

Desired Solution

```
channel_name => {
    conversations_list()
        .filter(c => c.name == channel_name)
        .map(c => { conversations_members(c.id)
                  .map(uid => { let u = users_info(user=uid)
                                    return u.profile.email })
            )
}]
```

Candidate #1

```
channel_name => {
    conversations_list()
        .filter(c => c.name == channel_name)
        .map(c => { conversations_members(c.id)
                  .map(uid => { let u = users_info(user=uid)
                                    return u.profile.email })
            )
}]
```

Candidate #2

```
channel_name => {
    conversations_open()
        .filter(c => c.name == channel_name)
        .map(c => { conversations_members(c.id)
                  .map(uid => { let u = users_info(user=uid)
                                    return u.profile.email })
            )
}]
```
Type-Directed Synthesis / Program Ranking

channel_name => {
  conversations_list()
  .filter(c => c.name === channel_name)
  .map(c => {
    let uid = c.creator
    let u = users_info(user=uid)
    return u.profile.email
  })
}

Desired Solution

channel_name => {
  conversations_list()
  .filter(c => c.name === channel_name)
  .map(c => {
    let uid = c.creator
    let u = users_info(user=uid)
    return u.profile.email
  })
}

Candidate #2
Program Ranking / Challenges in REST API Execution
Service providers set a rate limit
Program Ranking / **Challenges in REST API Execution**

- Service providers set a rate limit
- Many API calls have side effects
Program Ranking / Retrospective Execution

Desired Solution

```
channel_name => {
  conversations_list()
    .filter(c => c.name == channel_name)
    .map(c => {
        conversations_members(c.id)
            .map(uid => {
                let u = users_info(user=uid)
                return u.profile.email })
    })
}
```

Candidate #1

```
channel_name => {
  conversations_open()
    .filter(c => c.name == channel_name)
    .map(c => {
        conversations_members(c.id)
            .map(uid => {
                let u = users_info(user=uid)
                return u.profile.email })
    })
}
```

Candidate #2
Insight 3: replay from execution traces!

Desired Solution

```javascript
channel_name => {
  conversations_list()
  .filter(c => c.name == channel_name)
  .map(c => {
    let uid = c.creator
    let u = users_info(user=uid)
    return u.profile.email
  })
}
```

Candidate #2

```javascript
channel_name => {
  conversations_open()
  .filter(c => c.name == channel_name)
  .map(c => {
    conversations_members(c.id)
    .map(uid => {
      let u = users_info(user=uid)
      return u.profile.email
    })
  })
}
```
Program Ranking / **Retrospective Execution**

### Execution traces

```
...

let uid = channel_creator(c)

let u = users_info(user=uid)

...
```
Execution traces

Program

...  
let uid = channel_creator(c)  
let u = users_info(user=uid)  
...

**Program Ranking / Retrospective Execution**
Program Ranking / Retrospective Execution

Execution traces

Program

```
... let uid = channel_creator(c)
let u = users_info(user=uid)
...
```
Execution traces

Program

... 
let uid = channel_creator(c)

let u = users_info(user=uid)

...

exact execution
Program Ranking / Retrospective Execution

Execution traces

channel_creator

channel_creator

users_info

Program

... 

let uid = channel_creator(c)

let u = users_info(user=uid)

...

...
Execution traces

Program

...let uid = channel_creator(c)

...let u = users_info(user=uid)

...
Program Ranking / **Retrospective Execution**

**Execution traces**

```
let uid = channel_creator(c)
let u = users_info(user=uid)
```

**Program**

```
... approximate execution ...
```
channel_name => {
    conversations_list()
      .filter(c => c.name == channel_name)
      .map(c => {
        conversations_members(c.id)
          .map(uid => {
            let u = users_info(user=uid)
            return u.profile.email
          })
      })
}

Desired Solution

channel_name => {
    conversations_list()
      .filter(c => c.name == channel_name)
      .map(c => {
        let uid = c.creator
        let u = users_info(user=uid)
        return u.profile.email
      })
}

Candidate #1

channel_name => {
    conversations_open()
      .filter(c => c.name == channel_name)
      .map(c => {
        conversations_members(c.id)
          .map(uid => {
            let u = users_info(user=uid)
            return u.profile.email
          })
      })
}

Candidate #2
APIphany / Architecture

[1] Component-based synthesis for complex APIs. Feng et al. POPL’17
Evaluation / **Solved Benchmarks**

3 real-world APIs

32 benchmarks from StackOverflow and GitHub
Sample benchmark #1:

Task: Retrieve member emails from a channel name
Sample benchmark #2:
Task: Delete the payment source for a customer
Solves 29 out of 32 within 150s timeout
Evaluation / Spec vs Semantic Types

Solves 4 out of 32
Evaluation / **RE vs No RE**

Desired solutions ranked in **top 10** for **12** out of **32**
Evaluation / **RE vs No RE**

The graph compares the performance of models with and without RELABEL (RE) when evaluated on benchmarks. The x-axis represents the rank, while the y-axis shows the number of benchmarks. The graph includes a red region indicating that the number of benchmarks with RELABEL is lower than 1000.
Evaluation / **RE vs No RE**

Desired solutions ranked in top 10 for **23** out of 32
[1] Component-based synthesis for complex APIs. Feng et al. POPL’17