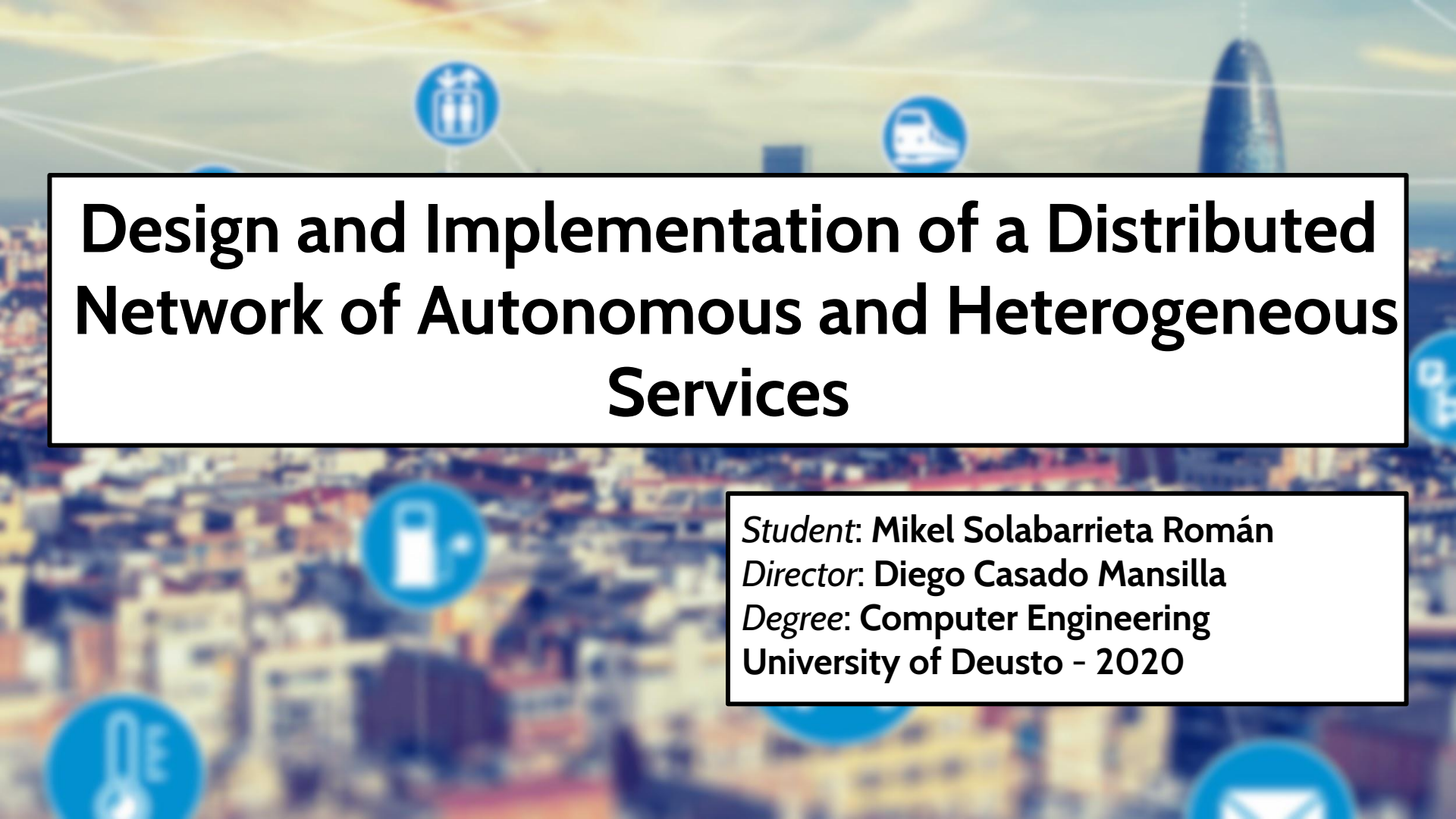


Design and Implementation of a Distributed Network of Autonomous and Heterogeneous Services

Student: **Mikel Solabarrieta Román**
Director: **Diego Casado Mansilla**
Degree: **Computer Engineering**
University of Deusto - 2020

Index

1. Introduction
2. BSPL
3. Networking
4. Demo
5. Results and Conclusions
6. Future Work
7. Questions and Answers

Introduction

Key areas (1/2)

Interaction-oriented design

Designing the system around agent interactions.

Separates interaction from implementation.

Information sent as messages but interpreted as interaction events.

Multi-agent systems

Composed of multiple independent agents.

Each agent has a local view of the system.

No single agent in charge, although control might be unevenly distributed.

Key areas (2/2)

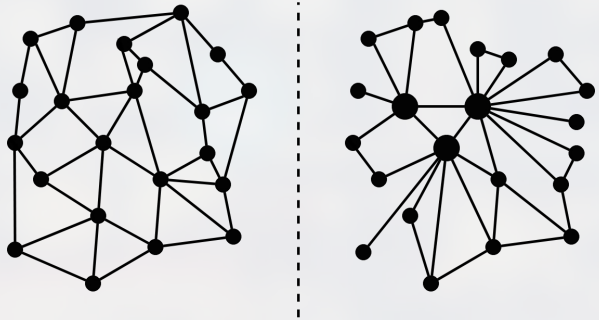
Peer-to-peer networks

Each node of the network can act as client and server.

Each node is “equal”.

Nodes connect directly, forming a distributed network.

Networking model mirrors logic model, but can face issues such as multiple NAT layers.



Goal of the project

Create a network of autonomous, heterogeneous services (NaHS).

Services are offered and consumed by agents.

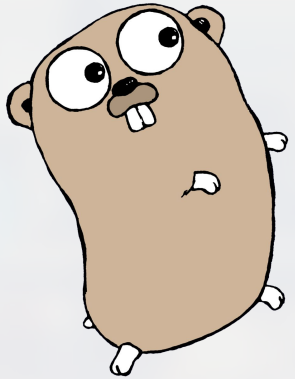
How agents interact is defined by protocols. Interacting is enacting a protocol.

An agent needs two things:

- Networking capabilities
- Interaction capabilities

Programming language

- Compiled.
- Statically typed.
- No classes
- Garbage collected.
- Native concurrency.
- Native test tools.
- Native package management.
- Pointers.
- Memory safety.




Blindingly Simple Protocol Language

What is BSPL?

Language to describe interaction protocols.

Designed with asynchrony in mind.

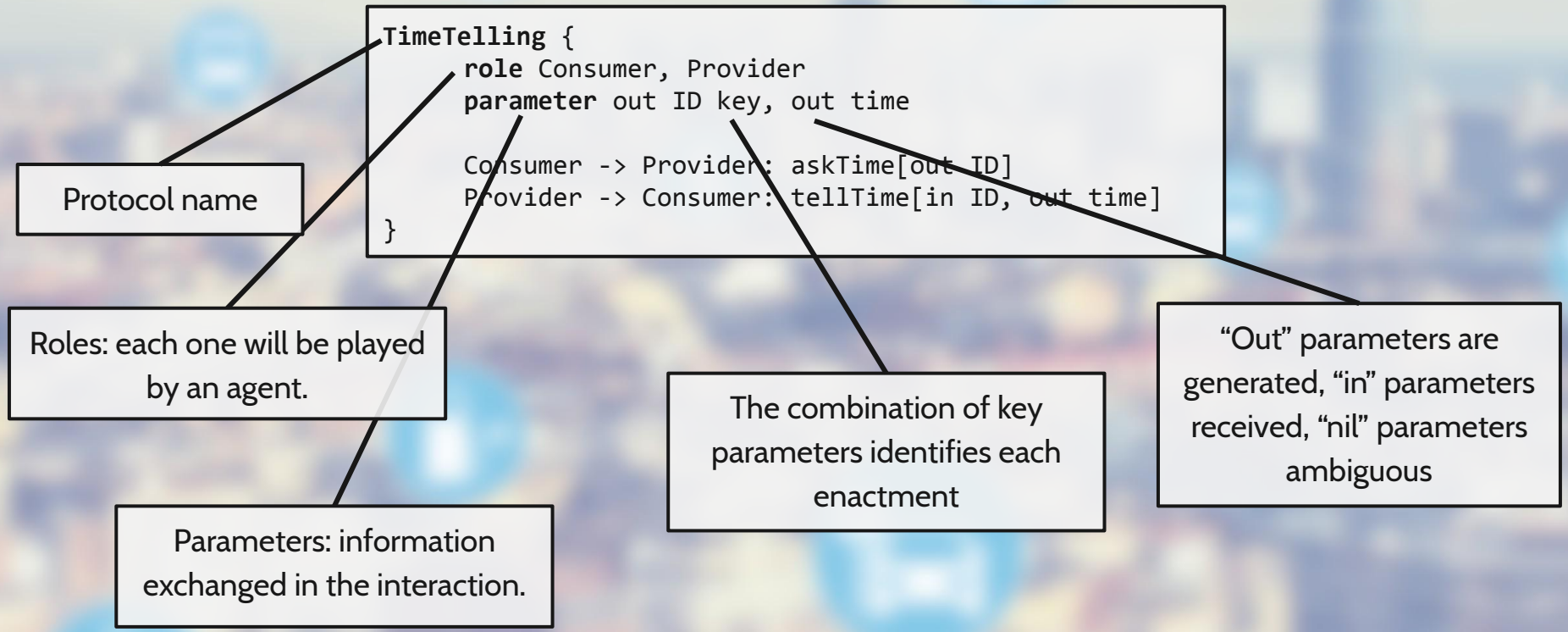
Contains a protocol name, roles, parameters and actions.

Causality is derived from parameters.

```
TimeTelling {
      role Consumer, Provider
      parameter out ID key, out time

      Consumer -> Provider: askTime[out ID]
      Provider -> Consumer: tellTime[in ID, out time]
}
```

What is BSPL?


TimeTelling {
role Consumer, Provider
parameter out ID key, out time
Consumer -> Provider: askTime[out ID]
Provider -> Consumer: tellTime[in ID, out time]
}
Protocol name
Roles: each one will be played by an agent.
Parameters: information exchanged in the interaction.
The combination of key parameters identifies each enactment
“Out” parameters are generated, “in” parameters received, “nil” parameters ambiguous

What is BSPL?

```
TimeTelling {
      role Consumer, Provider
      parameter out ID key, out time

      Consumer -> Provider: askTime[out ID]
      Provider -> Consumer: tellTime[in ID, out time]
}
```

Action = source + target + message

Message: new information in the enactment

Source: agent that started the action

Target: agent affected by the action

A parameter can't be received before being generated

The BSPL package

- **`proto`**
  Types and structures to define a BPSL protocol in Go.

- **`parser`**
  Converts raw protocols into the types defined by **`proto`**.

- **`reason`**
  Interfaces for protocol enactments (instances) and reasoners.

- **`implementation`**
  Implementation of all interfaces of **`reason`** except for the reasoner interface.

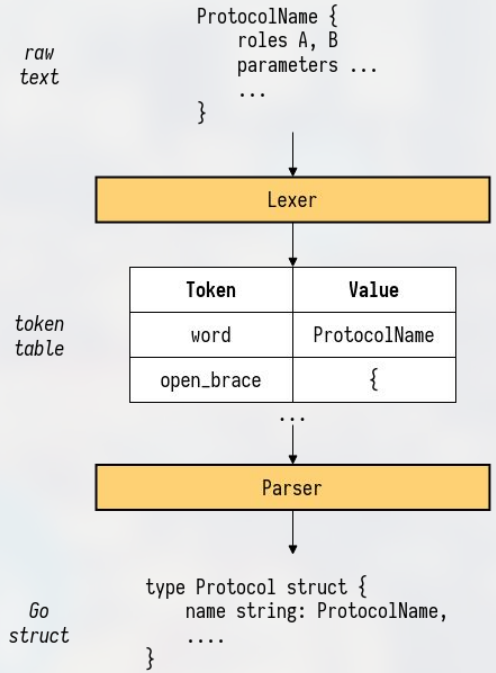
Parser

· Receives a raw protocol as input.

· Produces a Go representation of the protocol.

· Uses a lexer defined with a deterministic finite state automaton.

· Validates the basic correctness of a protocol.



Interfaces

Enable further development while maintaining implementation ambiguity.

```
// Instance of a Protocol
type Instance interface {
        // Diff identifies what action has been run between two versions of an
        // instance. It returns the action, the new values and an error.
        // Currently only one action is supported between instace versions.
        // An action slice is returned because two actions may have happened,
        // e.g. Accept or Reject. In that case the Reasoner must find out which
        // one it was.
        Diff(Instance) ([]proto.Action, Values, error)
        // Equals compares two instances.
        Equals(Instance) bool
        // GetValue returns the value of the parameter of an instance.
        GetValue(string) string
        // Key of the Instance.
        Key() string
        // Marshal an Instance to bytes.
        Marshal() ([]byte, error)
        // Parameters of the Instance.
```

```
// Reasoner handles the protocol instances and actions related to them
type Reasoner interface {
        // DropInstance cancels an Instance for whatever motive
        DropInstance(instanceKey string, motive string) error
        // GetInstance returns an Instance given the instance key
        GetInstance(instanceKey string) (Instance, bool)
        // All instances of a Protocol
        Instances(p proto.Protocol) []Instance
        // Instantiate a protocol. Check if the assigned role is a role
        // the reasoner is willing to play.
        Instantiate(p proto.Protocol, roles Roles, ins Values) (Instance, error)
        // RegisterInstance registers an Instance created by another Reasoner
        RegisterInstance(i Instance) error
        // UpdateInstance updates an instance with a newer version of itself
        // as long as a valid run from one to the other.
        UpdateInstance(newVersion Instance) error
}
```

Networking

LibP2P

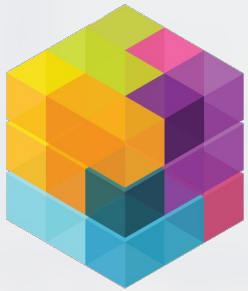
Modular network stack.

Addresses issues of peer-to-peer networking in different levels:
dialing, data transmission, identity, security...

Main implementations: JS and Go.

Host: type for network node.

Each host has a unique ID derived from its private key.



Discovery

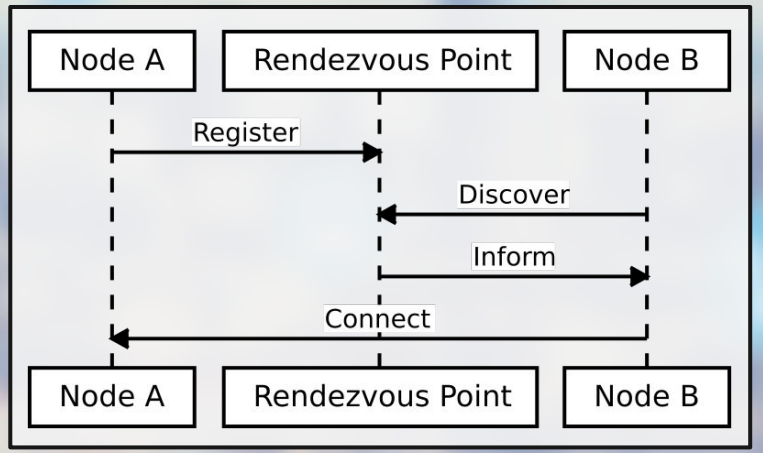
Rendezvous protocol.

Rendezvous nodes as initial connections.

Nodes register and are redirected to other nodes.

Nodes can also be manually added:

```
nodeB.Peerstore().AddAddrs(
      nodeA.ID(), nodeA.Addrs(),
      peerstore.PermanentAddrTTL
)
```



Design (1/2)

The **`Node`** type provides most of the networking capabilities:
discovery, event management, private networking...

Each agent has a **`Node`** attribute.

Each node has a LibP2P **`Host`** attribute.

`Node` has a **`Reasoner`** attribute, specified by each agent.

Private networks: by sharing a key.

Design (2/2)

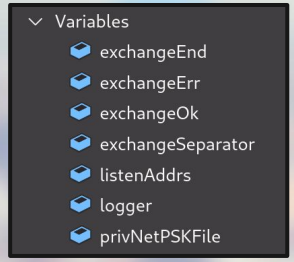
Events: marshalled instances that wrap interactions.

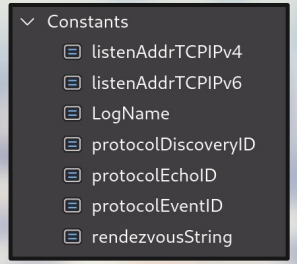
Events are sent with network streams via TCP connections.

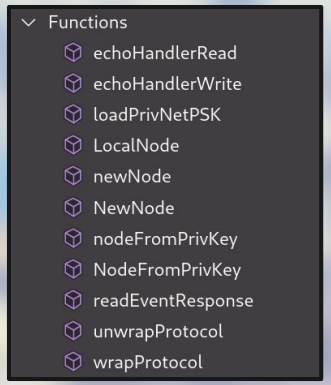
Flexibility by providing access to the LibP2P **`Host`**.

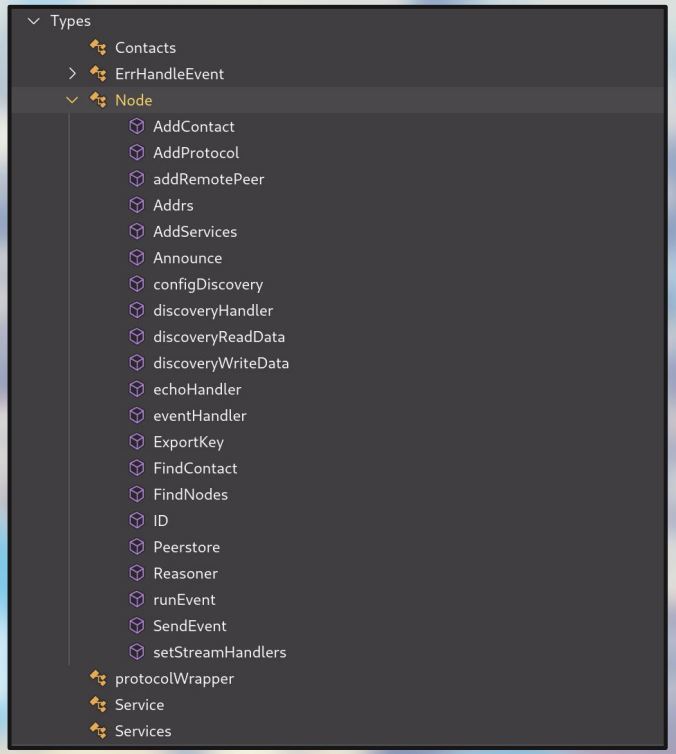
Accessibility by not requiring the user to access it.

Implementation









Functionality is centered around **`Node`**.

Images are from the **`net`** package.

Demo

Implementation

- Agent = reasoner + networking capabilities
- Each agent runs interactions as goroutines and can therefore continue functioning while they are processed.
- When necessary, different routines interact using channels.

```go
// Person is an agent representing human person
type Person struct {
        Node     *nahs.Node
        reasoner *personReasoner
}
```

```go
type personReasoner struct {
        Node *nahs.Node

        offeredServices  map[string]bspl.Protocol
        consumedServices map[string]bspl.Protocol
        openInstances    map[string]bspl.Instance
        droppedInstances map[string]bspl.Instance

        stationSearches map[string]chan string
        rentalRequests  map[string]chan string

        maxPrice        float64
        currentBikeRide string
}
```

```go
// Travel from src to dst
func (p Person) Travel(src Coords, dst Coords) error {

        // find nearest station
        logger.Infof("\t[%s] Search for station", shortID(p.ID()))
        result := make(chan string)
        errc := make(chan error)
        defer close(result)
        defer close(errc)
        go p.reasoner.stationSearch(src, result, errc)
        var station string
        select {
        case station = <-result:
                logger.Infof("\t[%s] Nearest station found: %s", shortID(p
        case err := <-errc:
                logger.Errorf("\t[%s] Couldn't find station: %s", shortID
```

Two versions (1/2)

Simple

Two agents: bike **rental** and **person**.

One protocol used to rent a bike.

Simple proof of concept.

Complex

More complex version of the simple demo, new agents expand the existing protocol and define new services with new protocols.

Six agents: bike **rental**, **person**, bike **station**, **transport** unit, **university** and **bicycle**.

Six protocols for: **renting** a bike, **scheduling** a rental, **transporting** bikes, **storing** bikes, **riding** bikes and **searching** for stations.

Two versions (2/2)

Simple

Highly deterministic: customer will always request a bike and rental will always respond.

Variability in accepting or rejecting the price.

Complex

Possibility of adding many agents that affect the outcome of the system interactions.

Many influential factors. Are there enough bicycles? Will there be at a certain our? Is the transport busy? ...

Some interactions trigger others. For example scheduling a bike rental may trigger a bike transport if there are currently no bikes available.

Design

```
BikeStorage {
        role Bike, Station
        parameter out ID key, in rentalID

        Bike -> Station: dock[ID key]
        Station -> Bike: release[ID, in rentalID]
}
```

```
BikeTransport {
        role Requester, Transport
        parameter out ID key, in bikeNum, in src, in dst, in datetime, out rID, out result

        Requester -> Transport: request[out ID, in bikeNum, in src, in dst, in datetime]
        Transport -> Requester: accept[in ID, out rID]
        Transport -> Requester: reject[in ID, out rID]
        Transport -> Requester: success[in ID, in rID, out result]
        Transport -> Requester: failure[in ID, in rID, out result]
}
```

Results and Conclusions

Results: BSPL

README.md

Blindingly Simple Protocol Language (BSPL) Go parser.

build passing | codecov 74% | License MPL 2.0 | Go v1.14 | godoc reference

This repository also contains interfaces for a BSPL reasoner (`reason` package) and an implementation of some components of that reasoner (`implementation` package). This implementation is used in another project.

Modules

- `parser` : Standalone BSPL parser implemented using a toy lexer I wrote a while ago.
- `proto` : Go structures to form a BSPL protocol, e.g., `Protocol` , `Role` and `Action` .
- `reason` : Interface definition for implementing a reasoner and protocol instances.
- `implementation` : Draft implementation to use in another project.

Production use of this project is not advised as it is far from ready.

Other folders

- `config` : Contains the automaton fed to the lexer to process a BSPL protocol.
- `test` : Test resources.

Usage example

1. Define a valid protocol in a file with path `path` .

github.com/mikelsr/bspl

package bspl v0.0.0 (50b84e0) Latest

Published: **Jun 18, 2020** | License: MPL-2.0 | Module: github.com/mikelsr/bspl

Doc | Overview | Subdirectories | Versions | Imports | Imported By | Licenses

Index

Index

func Compare

```
func Compare(a, b Protocol) bool
```

Compare two BSPL protocols

type Action

```
type Action = proto.Action
```

Action is an alias for proto.Action

Results: NaHS

README.md

NaHS

Network of Autonomous and Heterogeneous Services (NaHS)

build passing | codecov 61% | License MPL 2.0 | Go v1.14 | godoc reference

Modules

- `events` : Describes BSPL instance events according to the implementation. As of now there are three events:
  - `NewEvent` to create an instance of a protocol.
  - `UpdateEvent` to update an instace comparing it to a future version of it.
  - `DropEvent` to cancel an instance for any reason.
- `net` : Networking components. The main struct is `Node` . A node has a BSPL reasoner and a LibP2P host, implementing methods and handlers to send BSPL components between network peers. Nodes discover each other either manually or with the libp2p implementation of rendezvous (**preferred**) using the default bootstrap nodes.

Other folders

- `config` : Contains the private key of the main network (which is public, private only limits interaction to NaHS nodes).
- `scripts` : Contains a script to generate a private network key.
- `test` : Test resources.

github.com/mikelsr/nahs

package nahs v0.0.0 (c517c0f) Latest

Published: **Jun 18, 2020** | License: MPL-2.0 | Module: github.com/mikelsr/nahs

Doc | Overview | Subdirectories | Versions | Imports | Imported By | Licenses

Index

Index

type Node
  func MakeNode(reasoner bspl.Reasoner, sk crypto.PrivKey, options ...libp2p.Option) *Node
  func NewNode(reasoner bspl.Reasoner, options ...libp2p.Option) *Node

type Node

```
type Node = net.Node
```

Node of the NaHS network.

func MakeNode

```
func MakeNode(reasoner bspl.Reasoner, sk crypto.PrivKey, options ...libp2p.Option) *Node
```

MakeNode creates a node with the specified private key so the node maintains the ID it previously had.

func NewNode

```
func NewNode(reasoner bspl.Reasoner, options ...libp2p.Option) *Node
```

NewNode creates a new NaHS node. LibP2P options can be passed to configure the node.

Results: Usage

```go
package main

import (
        "fmt"
        "os"

        "github.com/mikelsr/bspl"
)

func main() {
        source, _ := os.Open(path)
        protocol, err := bspl.Parse(source)
        if err != nil {
                panic(err)
        }
        fmt.Println(protocol)
}
```

```go
package main

import (
    "fmt"

    "github.com/mikelsr/bspl"
    "github.com/mikelsr/nahs"
)

func main() {
    var reasoner bspl.Reasoner
    node := nahs.NewNode(reasoner)
    fmt.Printf("Created a new NaHS node with ID: %s\n", node.ID())
}
```

Results: Demo

```
Created bike with ID LLAQ (QmcaLXdAC1RWfGJvyeZ6MugBEqoDSXVgfXh3x6RUD9LLaQ)
Created bike with ID XWWJ (QmPftWe8DNkt7HqEN6kAmD6njWM3dar7XJA6bnujqkxWwJ)
Created bike with ID WEAF (QmYJb9ovv1LbAwuocv7roxP9VzTrjZZuUFJLLTgoL5weaf)
Created bike with ID FCNS (QmQFUixwJM8XychHHfPtMn5o9x4yLJCcZbKexkfPVnFCNs)
Created station with ID BXGE (QmRSZSGMnkbGkj9MzippGrKZoChBMdLiHGLgeQ666JBxgE)
[BXGE] Bike LLAQ docked
[BXGE] Bike XWWJ docked
Created station with ID XUC8 (QmRcQxF3nWPEZLBNrWUoW8sWX1TpVBczEDHLtayBpmxuc8)
[XUC8] Bike WEAF docked
[XUC8] Bike FCNS docked
Created transport with ID PQUB (QmX1gr24SNdYwEnvR5KwvRJ8CShQdMSozURgyK2y81PqUB)
Created university with ID 7BTP (QmXDcZLmytcWGzciWLSHAxc5ZKTJeG1NZe4Kz8DCXd7bTp)
Created renter with ID W7GW (QmVMhuBdVxtsm3pykK4fbgtS92sLE9oAbiE1FJstKCw7gw)
Created person with ID ABZJ (QmPqcXKsudZ86Py2iFHNRfeAHK1ck19MnM3bFuqEGkABZJ)
[ABZJ] Search for station
[W7GW] Send event 'update:4261' to node ABZJ (instance key: StationSearch,ID:dccd7a0a-b791-4f2a-a41f-99d7ea8575f5)
[ABZJ] Nearest station found: BXGE
[ABZJ] Sent rent request to W7GW
[W7GW] Send event 'update:D951' to node ABZJ (instance key: BikeRental,ID:708d84b4-6017-42eb-842c-42429ebc96ee)
[ABZJ] Received offer for bike LLAQ at price: '0.03'
[ABZJ] Accepted offer for price '0.03'
[ABZJ] Bike with id LLAQ rented
[ABZJ] Waiting to discover node LLAQ
[ABZJ] Send event 'update:9585' to node W7GW (instance key: BikeRental,ID:708d84b4-6017-42eb-842c-42429ebc96ee)
[ABZJ] Discovered LLAQ
[ABZJ] Send event 'new:0FED' to node LLAQ (instance key: BikeRide,ID:f4f48beb-3f82-4f2b-8e75-6df9744d0dc0)
[W7GW] Send event 'update:C64D' to node ABZJ (instance key: StationSearch,ID:ff0ba9fb-150f-4b4c-a9c2-4371ad2675a0)
[W7GW] Response from ABZJ for bike LLAQ offer: accept
[ABZJ] Nearest station found: XUC8
[ABZJ] Dropping bike LLAQ at station XUC8
[7BTP] Requesting 2 bike(s) from W7GW to station BXGE at 2020-06-18 23:53:00.364080301 +0200 CEST m=+3.593868373
[LLAQ] New rider ABZJ
[ABZJ] Send event 'update:ED90' to node LLAQ (instance key: BikeRide,ID:f4f48beb-3f82-4f2b-8e75-6df9744d0dc0)
[LLAQ] Dropped at XUC8 by ABZJ
```

Conclusions

Project reached its goal and met the objectives.

Agents can interact by enacting protocols that describe services. Agents form a distributed network with peer-to-peer connections.

Agents: use BSPL and NaHS packages, information driven, can be asynchronous.

It provides an initial approach to networks of autonomous and heterogeneous services.

Future Work

Future Work

Design and implement an accountability and trust system.
What happens when a node consistently breaches trust? How can other nodes know what nodes tend to behave better? How to choose between multiple providers?

Expand the features of the networking package to make agent design easier.
Remove circular reference, provide a more intuitive way of running callback functions when updating events.

Deploy NaHS in a real scenario.
The bicycle scenario proposed in the demo is a valid target with services such as BilbaoBizi.

Q&A