Lab #3:
KP development with SmartSlog

Instructor: Ivan Galov

Why SmartSlog?

Advantages

- Oriented to low-performance devices
- Support multilingual ontology library generation
- Developer thinks in terms of ontology entities
- Mechanism of traffic reducing
- Complex queries
- Optimizations
- Ontology manipulations
- Based on C_KPI

SmartSlog

1. **API**

   See file smartslog.pdf

2. **Basic scenarios**

- Initialize connection to Smart Space
- Send individual to Smart Space
- Receive individual from Smart Space
- Search individual by pattern
- Subscription to individual's properties
- Asynchronous subscription

SmartSlog: installation

- Download SmartSlog (SmartSlogKPlib_ANSI-C.tar.gz)
- Unpack

```
~> tar -zxf SmartSlogKPlib_ANSI-C.tar.gz
~> cd SmartSlogKPlib_ANSI-C
```

- Install

```
~> ./configure --prefix=$HOME/inst
~> make
~> make install
```

SmartSlog: example

Example

- Download media example (media_example.tar.gz)
- Unpack

```
~> tar -zxf media_example.tar.gz
~> cd media_example
```

- Edit SIB IP and port in *player/player.c* (line 29) and *phone/phone.c* (line 49):

```
sslog_ss_init_session_with_parameters("X", "127.0.0.1", 10010);
```

- Compile example

```
~> make
```

- Run (in different terminals)

```
~> ./player
~> ./phone
```

SmartSlog: Initialize connection

```
sslog_ss_init_session_with_parameters("X",
                                      "127.0.0.1", YOUR_PORT);
register_ontology();

if (ss_join(sslog_get_ss_info(), "KP name") == -1)
{
        /* Error handle */
}

/*
... KP logic ...
*/

ss_leave(sslog_get_ss_info());
sslog_ss_leave_session_all();
sslog_repo_clean_all();
```

SmartSlog: Send individual

```
individual_t * phone = sslog_new_individual(CLASS_MOBILEPHONE);

/* insert individual with manual UUID */
sslog_ss_init_individual_with_uuid(phone, «phone1»);

sslog_add_property(phone, PROPERTY_CURRENTVOLUME, «value»);
...

if (sslog_ss_insert_individual(phone) < 0) {
                        /* error */
}
```

SmartSlog: Receive individual

```
individual_t * phone =
sslog_new_individual(CLASS_MOBILEPHONE);

/* set uuid of individual you want to receive */
sslog_ss_init_individual_with_uuid(phone, «uuid1»)

/* populate to local store */
prop_val_t *property= sslog_ss_get_property(phone,
                                            CLASS_MOBILEPHONE);
/* ... */
```

SmartSlog: Search individual

```
individual_t *pattern =
                              sslog_new_pattern(CLASS_MOBILEPHONE);
list_t * individuals = NULL;

/* set properties for individual you are searching */
sslog_add_property_to_pattern(pattern, ...);
sslog_add_property_to_pattern(pattern, ...);
/* … */

/* populate to local store */
individuals = sslog_ss_get_individual_by_pattern_all(pattern);

if (individuals == NULL)
{ /* Unable to query */ }

free_individual(pattern);
```

SmartSlog: Subscription

```
individual_t *phone = sslog_new_individual(CLASS_MOBILEPHONE);

subscription_t *subcription = sslog_new_subscription(false);

/* Prepare to subscription */
list_t *properties = list_get_new_list();
list_add_data(PROPERTY_VOLUME, properties);
sslog_sbcr_add_individual(subcription, phone, properties);

/* Subscribe to defined properties */
if (sslog_sbcr_subscribe(subcription) != 0)
{ /* Handle error */ }

while (true) {
        const prop_val_t *p_val = sslog_get_property(phone,
                                        PROPERTY_ISCALLING->name);
        /* New value in p_val. Working with new properties */
        sslog_sbcr_wait(subcription);
}
```

Task

- Use **“HelloWorld”** ontology
  (already generated, download helloworld.tar.gz)

- **First KP** should initialize whole individual and publish some properties
- **Second KP** should subscribe to published properties

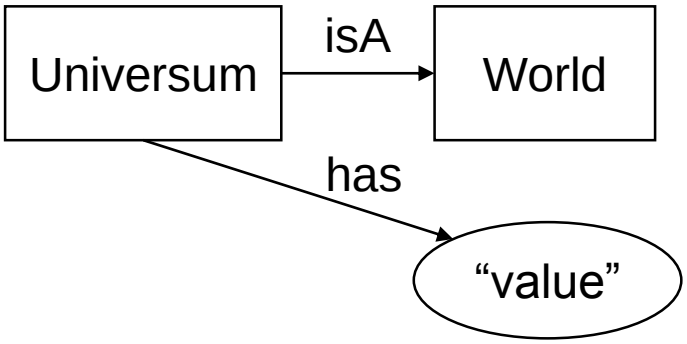
Optional:

- Extend Second KP to **asynchronous** subscription

HelloWorld ontology

CLASS_UNIVERSUM;
CLASS_WORLD;

PROPERTY_ISA;
PROPERTY_HAS;



Use CLASS_UNIVERSUM and PROPERTY_HAS (data-property).

Publisher:
1) creates an individual from universum class with some uuid;
2) manipulates with the 'has' property;
3) Inserts individual to the smart space.

Consumer:
1) creates the individual from universum class with the same uuid;
2) subscribes to the 'has' property
3) receives new data by subscription.