Universal Meaning Representation Format (UMRF) for Natural Language Task Engines

Presenters: Robert Valner (University of Tartu)  
Selma Wanna (University of Texas)
Motivation and Demonstration

- The **Universal Meaning Representation Format (UMRF)** - generalizes third-party meaning representations to a common formalism, making them accessible to **Robot Task Management Systems**

- **Implications:**
  - Separates **parsing** and **task execution** layers in Natural Language (NL) pipelines
    - Easier to swap parsers
    - Accelerates development and testing of NL systems
  - Flexible enough to **represent many input modalities**
  - Promotes corobot applications by providing **better interfaces for developing human-robot interaction systems**

- **Demo materials available at**
High-Level Overview: Front End

Audio input is processed through a microphone, which sends speech to text. The text is then converted to a JSON request that is sent to the UMRF. The cloud service processes the request and the ROS node receives the command to move Jack forward.

- "Jack move forward"
- "Alexa, move Jack forward"
- "Ok Google, move Jack forward"

The JSON request includes a "verb" and "direction" field, such as "verb": "move" and "direction": "forward".
High-Level Overview: Back End

- Action Engine
  - Action Match Finder
  - Action Indexer
  - Action Executor
    - load Action
    - instantiate Action
    - execute Action

- compiled Actions (shared libraries)
- indexed Actions

- “verb”: “move”
  - “direction”: “forward”
UMRF Graph

- **UMRFs can build upon each other** to invoke more complex behaviour
  - Parallelism
  - Cycles

- **Parameters** can be passed along **Actions**
  - Default data types (strings, numbers, boolean)
  - Custom data types (objects, containers, pointers)
UMRF JSON Syntax

- Design founded in predicate-argument semantics with influence from slot-intent Meaning Representations (MR)

- UMRF Data Fields
  - **Name** - name of the action
  - **Input Parameters** - input information for the action
  - **Output Parameters** - the resulting information after the action is performed

```json
{
  "name": "NavigateTo",
  "package_name": "ta_navigate",
  "input_parameters": {
    "verb": {
      "pvf_type": "str",
      "pvf_val": "navigate"
    },
    "location": {
      "pvf_type": "str",
      "pvf_val": "kitchen"
    }
  },
  "output_parameters": {
    "goal": {
      "pvf_type": "geometry_msgs::Pose"
    }
  }
}
```

Generated UMRF JSON for “Robot, go to the kitchen”. The output data can be used by other actions when combined in graph.
Developed ROS tools

- **TeMoto Action Engine**
  - Implements UMRF Graph execution back-end in C++
  - Freely available at [https://github.com/temoto-telerobotics/temoto_action_engine](https://github.com/temoto-telerobotics/temoto_action_engine)
  - Apache 2.0

- **TeMoto Action Assistant**
  - GUI tool for creating base for TeMoto Actions
  - Freely available at [https://github.com/temoto-telerobotics/temoto_utils](https://github.com/temoto-telerobotics/temoto_utils)
  - Apache 2.0
The bigger picture - TeMoto framework

- **TeMoto framework** is a set of ROS based tools that help to **rapidly develop** semi-autonomous teleoperated systems

- **Actions** utilize the **Managers** via resource queries,
  - e.g., Component Manager → *start the camera* …

- **Actions** keep the application code **modular and scalable** and **Managers** provide **resource abstraction**, **dynamic allocation** and **simple API**

More information on [temoto-telerobotics.github.io](http://temoto-telerobotics.github.io)
Conclusion & Future Work

● **UMRF is an intuitive convention** that allows to
  ○ merge different NLP systems
  ○ segregate front-end interfaces from back-end task management

● **TeMoto Action Engine** is C++ implementation of UMRF definitions

● **Future Work:**
  ○ Extensive testing on more MRs and more diverse task spaces
  ○ **Simple developer tools** for designing and testing **UMRF Graphs**
  ○ Creating a tool that maps slot-intent MRs to the UMRF (automatic conversion)
  ○ Create an open-source Temoto parser based on task grammars