

# Progress in Natural Language and Knowledge Representation

David Devault (ICT), Arno Hartholt (ICT), Eduard Hovy (ISI), Anton Leuski (ICT), Tom Russ (ISI), David Traum (ICT) | 9/24/2008

(presented by Eduard Hovy)



The projects or efforts depicted were or are sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM), and/or the US Army Research Institute. The content or information presented does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.



# Even VHUMANs mature...

---

- **MRE and SASO: We have built a very sophisticated VHUMAN:**
  - Most innovative design and most complex internal structure in the research literature
  - Powerful capabilities of each module plus rich internal interconnections provide system's flexible and reactive behavior
- **But: the system has been experimental and exploratory**
- **Recently, our work has started to mature...**
  - We want to be able to **extend** the system (e.g., add new agents) **rapidly**
  - We want to **create new scenarios** easily
  - We would like to **enable other people** to build scenarios
  - Many of our **modules** are beginning to be **used in other projects**
- **This is typical R&D evolution: idea → pilot → prototype → re-use → distribution**
- **So... We are re-engineering and centralizing much of the infrastructure to make cross-module growth and changes easy and consistent**

# Current state of affairs

---

- **Problem: A Babel inside the system:**

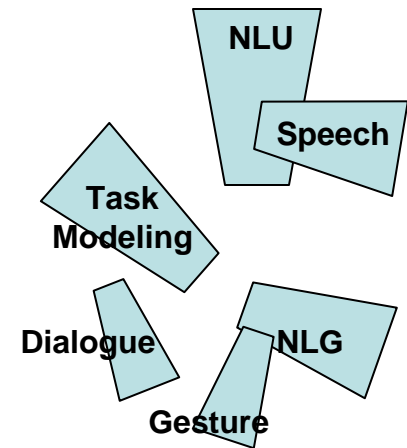
- $N$  major modules ... each performing a different function
- BUT each has its own different internal representation terms and modeling style
- Why?
  - We started with some advanced legacy systems — building everything fresh would have taken too long
  - Each module is pushing the state of the art in its own research field — there's no reason their representations and methods should correspond

- **This makes it a LOT of work to extend functionality that crosses modules:**

- For example, you can add new words to NLU, but that doesn't mean that NLG knows the words!

- **Desiderata for 'one-shot extensibility'**

- Should be easy to build new knowledge
- New additions should be made consistent, also with existing knowledge
- Knowledge should be compositional: new knowledge should plug into older knowledge
- Knowledge should be used by all modules in the same ways



# Example: Extending the system

---

- **Want to add new line of negotiation:**
  - EXISTING: Doctor can negotiate for delivery of medical supplies downtown
  - NEW: Want Elder to negotiate delivery of power generator downtown
  - So: Elder needs to know
    - Task model module: Define “deliver” action, its preconditions, effects, dependencies on other entities:
      - Operating the clinic downtown requires power
      - Having power downtown would be a good thing
      - Precondition for power is a power generator
      - There isn’t one today, so it has to be obtained and delivered
      - etc.
    - NLU and NLG and Speech modules:
      - Lexicons: Words for “power generator”, “deliver”, “need”, etc.
      - Grammars: Patterns and phrases using these words that can accommodate the negotiation
      - Semantics: Representation frames that represent these concepts
- **Currently this is hard to do: even though the Doctor knows all this, the Elder doesn’t, and his various modules must *each* be extended**

# Example: Adding new knowledge (old method)

NLU: FRAMEBANK

<S> we can deliver power generators </S>

NLG: FRA

NLG: FRAMER

NLU: FRAMEBANK

<S> th

<S> we can not provide power generators </S>

addre

dialog

NLU: FRAMEBANK

dialog

<S>

dialog

<S> we can provied a power generator </S>

speed

<S>

speed

<S>.sem.event deliver

speed

<S>.sem.deliver.polarity negative

speed

<S>.sem.source us-army

speed

<S>.sem.agent captain-kirk

speed

<S>.sem.theme power-generator

speech-act.actor elder-al-hassan

power-generator }  
-power-generator(0.95)  
electricity(0.95) }

town

# Example: Adding knowledge for NLU

---

- **Of course we create tools and interfaces to speed up system extension**
- **Still, it can be a lot of work to add the knowledge needed for NLU:**
  - Define the words
  - Define the representation frame elements
  - Link them together
  - Ensure that they integrate with everything else

QuickTime™ and a  
H.264 decompressor  
are needed to see this picture.

# Today: Extending NLG, using auto-generation

---

- **Evolved specialized procedure to facilitate extensions:**
  - Builder adds knowledge for one new phrase
  - System automatically creates variations, using grammar
- **For NLG, sequence is:**
  - Want to say “power generator”
  - Generator can’t produce sentence
  - Addition interface: provide needed knowledge (yes, it’s ugly!)
  - Back to NLG: now produces sentence
  - ...AND: its auto-extension produces six more variations for free!
  - BUT: this is not yet centralized, not exported to NLU...

# Extending NLG: Before addition

---

QuickTime™ and a  
Video decompressor  
are needed to see this picture.



# Extending NLG: Making addition

---

QuickTime™ and a  
Video decompressor  
are needed to see this picture.

# Extending NLG: After addition

---

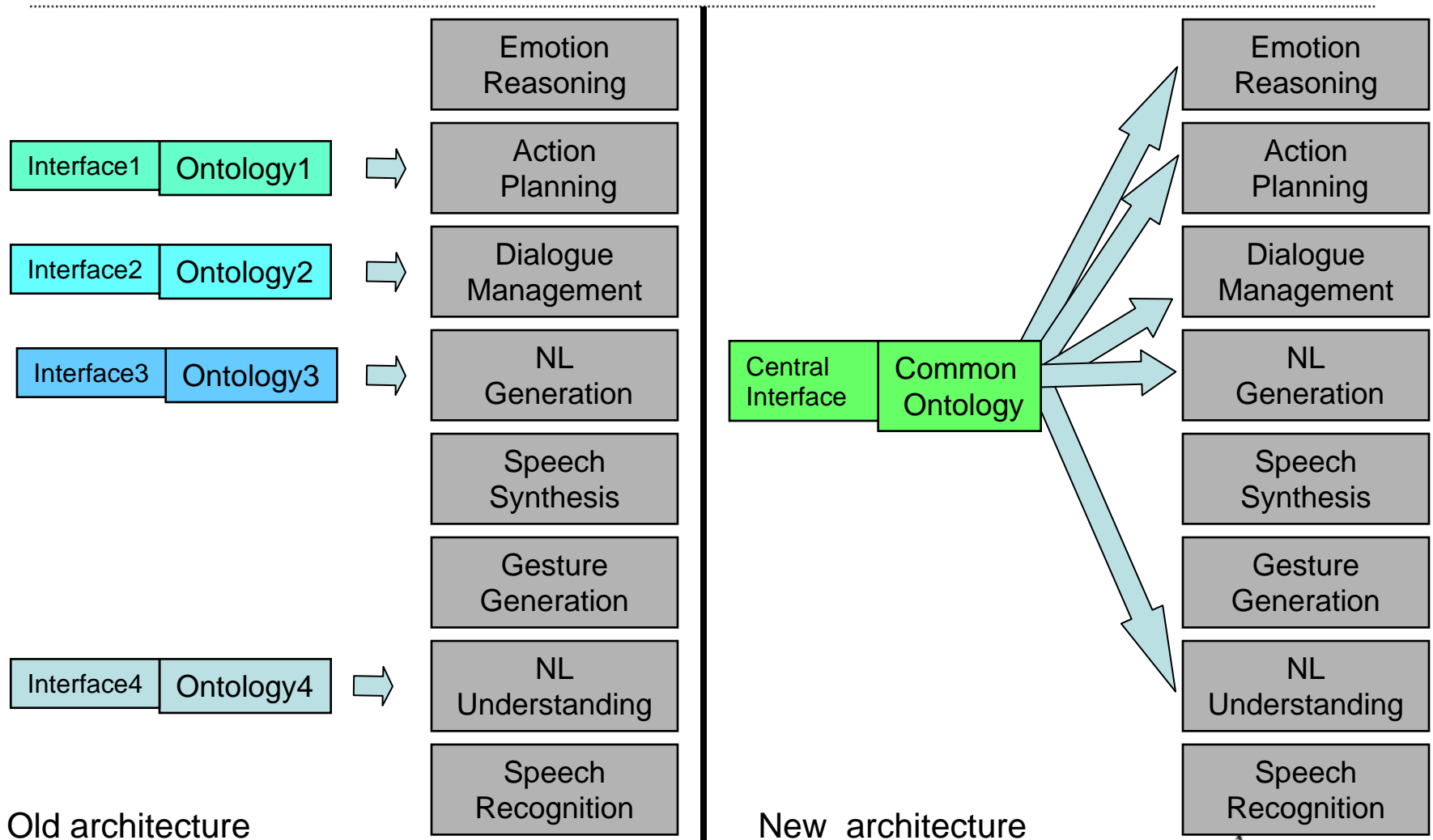
QuickTime™ and a  
Video decompressor  
are needed to see this picture.

# Where are we going? A centralized framework

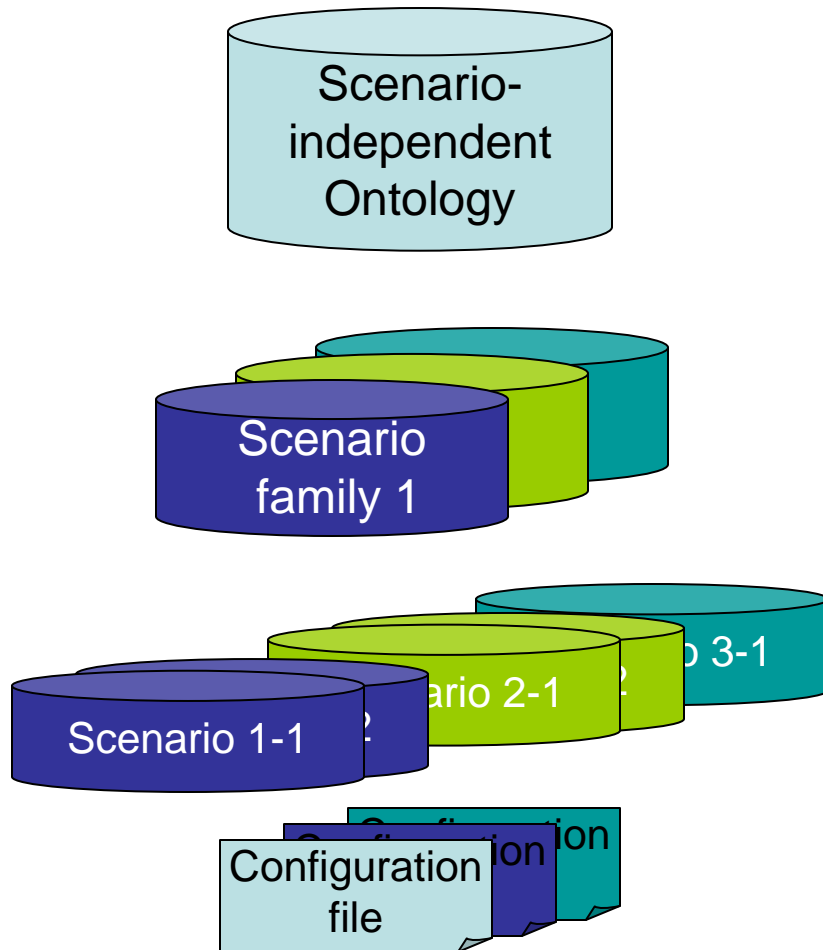
---

- **A single central ontology of terms:**
  - Standardizes all the terms used by all modules
  - Makes explicit most of the detailed knowledge currently held by experts
- **A single content-building/checking interface (Protégé, from Stanford):**
  - Provides a single point of entry of all necessary info, all related
  - Supports a scripted series of content-building steps, during which the interface requests different related kinds of info (as needed for each module)
- **Automated internal consistency checking**
- **Automated content enhancer/exporter functions:**
  - Automatically inherits required info from ontology and fleshes out new input material
  - Automatically converts fleshed-out input into formats used by various modules

# Old and new architecture



# New ontology organization



- General world knowledge
  - (generic actions, objects...)
- Linguistic structures
- Generic dialogue items

- Scenario actor classes
- Scenario action templates
  - (generic preconds, effects...)
- Propositions for scenario

- Specific actors, locations, etc.
- Sentences for scenario

- Actor positions, attitudes, etc.
- Simulation initialization

# Ontology specifies action templates

The screenshot shows the Protégé 3.3.1 interface. The main window is titled "saso-en-scenario Protégé 3.3.1" and shows the "CLASS EDITOR" for the class "world:DeliverAction". The "SUBCLASS EXPLORER" on the left shows a hierarchy of classes under "world:Action", with "world:DeliverAction" selected. The "CLASS EDITOR" displays the "Property" tab for "world:DeliverAction", showing a table with "rdfs:comment". Below the table, the "Asserted Conditions" section shows a list of conditions for "world:DeliverAction", including "world:actionVerbFamily has world:deliverVerbFamily", "world:addList has destination.resourceAttribute Equal theme.", and "world:precondition has source.resourceAttribute Equal theme.". A callout box points to the "world:DeliverAction" class in the hierarchy, and another callout box points to the "world:actionVerbFamily has world:deliverVerbFamily" condition in the "Asserted Conditions" section. A third callout box points to the "world:DeliverAction" class in the "Asserted Conditions" section.

File Edit Project OWL Code Tools Window SASO Change Help

Metadata (saso-en-scenario) OWLClasses Properties Individuals Forms SASO Tools Changes Change Statistics

**SUBCLASS EXPLORER**  
For Project: saso-en-scenario

Asserted Hierarchy

- owl:Thing
  - rdfs:Class
  - rdf:Property
  - world:CaseRoleProxy
  - world:Color
  - world:Entity
  - world:Event
  - world:Action
    - world:ActionReducingNeutrality
    - world:AgreeAction
    - world:AttackAction
    - world:BuildAction
    - world:DeliverAction
    - world:GiveAction
    - world:MoveAction
    - world:OperateFacilityAction
    - world:ProtectAction
    - world:ProvidePublicServicesAction
    - world:RenovateAction
    - world:SupportAction
    - world:TellAction
    - world:TreatAction

**CLASS EDITOR**  
For Class: world:DeliverAction

Property	
rdfs:comment	

Asserted Conditions

- world:Action
- world:actionVerbFamily has world:deliverVerbFamily
- world:addList has destination.resourceAttribute Equal theme.
- world:precondition has source.resourceAttribute Equal theme.

Logic View Properties View

Templates define semantics of action:  
*Ex: One effect of a deliver action is that the destination receives the thing being delivered (the 'theme')*

Hierarchy allows inheritance of semantics

# Creating a new *deliver* action

2. Create instance

3. Add details

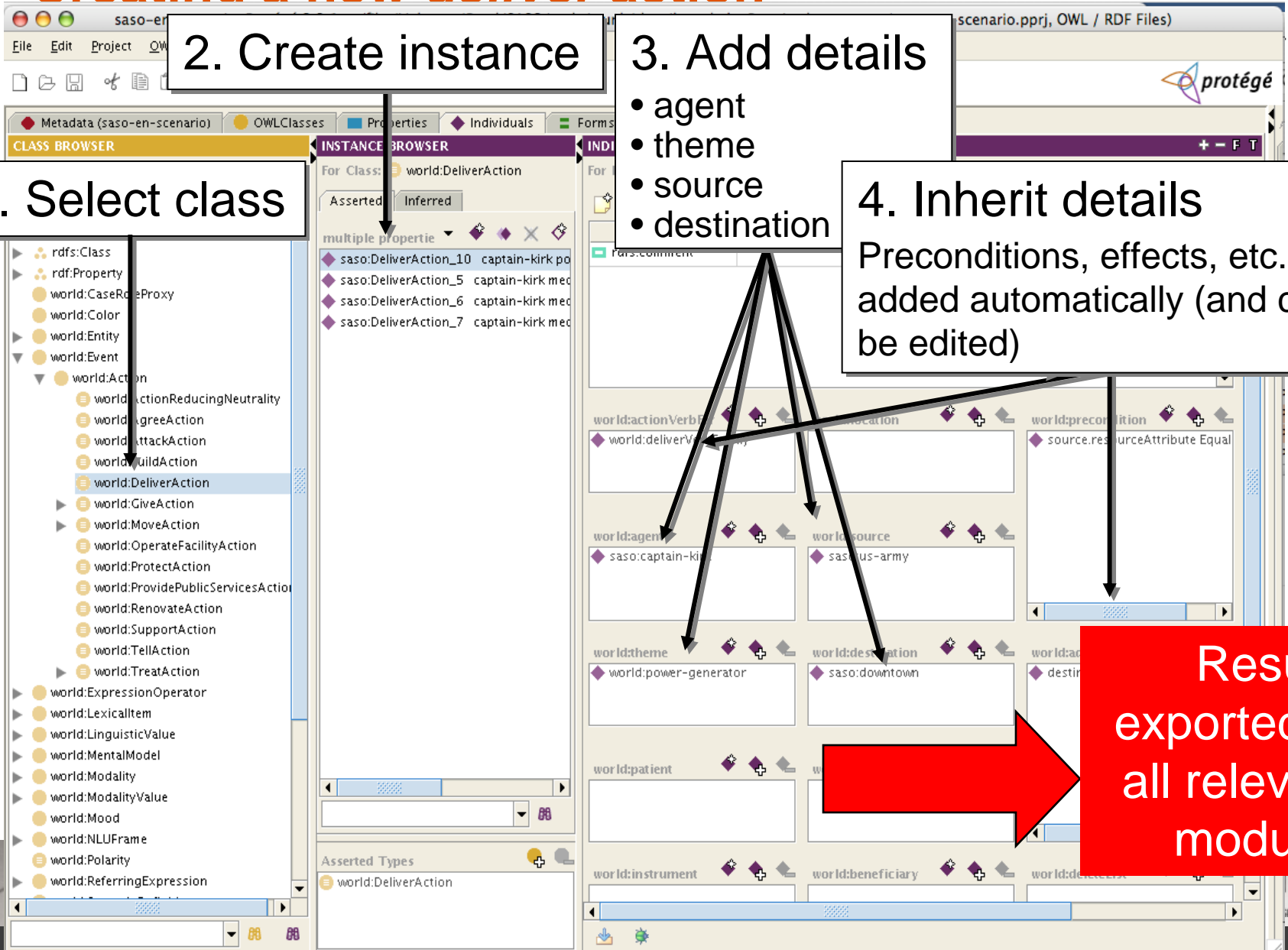
- agent
- theme
- source
- destination

1. Select class

4. Inherit details

Preconditions, effects, etc. are added automatically (and can be edited)

Results exported to all relevant modules



# Templates simplify adding *deliver*

## Old method

Create new Deliver action instance  
Specify agent = "CaptainKirk"  
Specify theme = "power-generator"  
Specify source = "us-army"  
Specify destination = "downtown"

Create state "downtown-has-power-generator"  
Create task model "downtown-has-power-generator"

Specify belief  
Specify initial-value  
Create state "us-army-has-power-generator"  
Create task model "us-army-has-power-generator"  
Specify belief  
Specify probability  
Specify initial-value

Create new task instance  
Specify ground event is the deliver action instance  
Specify precondition "us-army-has-power-generator"  
Specify add-effect "downtown-has-power-generator"

## New centralized method

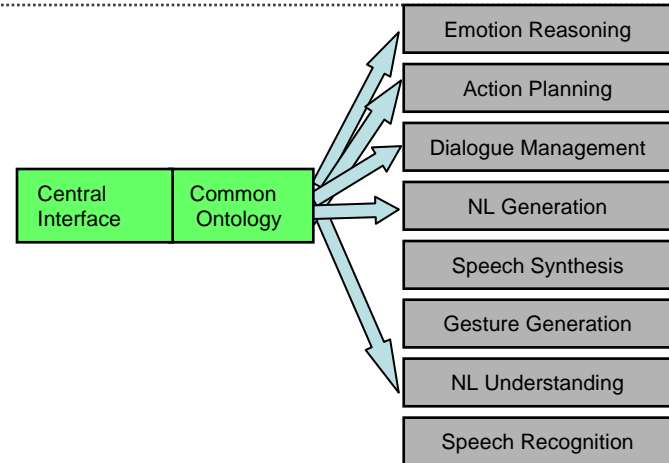
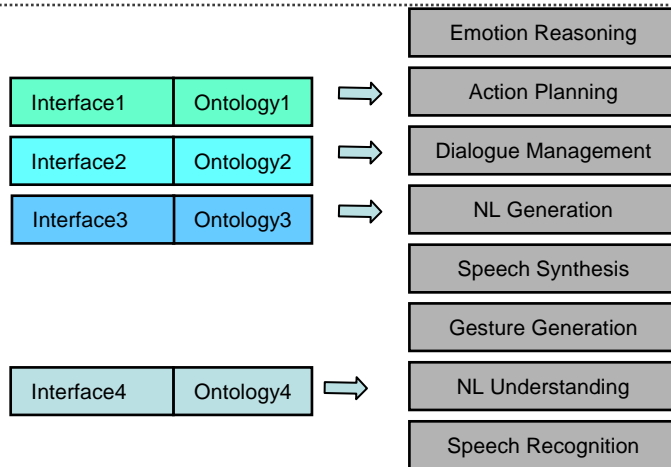
Create new Deliver action instance  
Specify agent = "CaptainKirk"  
Specify theme = "power-generator"  
Specify source = "us-army"  
Specify destination = "downtown"

Create state "downtown-has-power-generator"  
Create task model state  
    "downtown-has-power-generator"  
Specify belief

Create new task instance  
Specify ground event is the deliver action  
instance



# Work required: The old and the new



## ▪ In the past: system builder

- identify core problem,
- locate problem in each module,
- design all fix/extension(s),
- implement change(s) in each module,
  
- coordinate change across modules,
- perform extensive tests,
- perform cross-module debugging.

## ▪ In the future: system builder

- identify core problem,
- design fix/extension,
- implement change in central repository, letting tools perform consistency check
  
- perform final tests and one-shot centralized debugging.

# Where next?

---

- **We're partway through the work...**
  - Completed central standard ontology
    - Using Stanford's Protégé for content building
    - Using Protégé's interface for ontology access/browsing
    - Using OWL (Semantic Web) representation formalism
    - Used some concepts from various upper-level ontologies, such as SUMO and Omega
  - Completed exporter functions to some modules
- **To be done in coming year:**
  - Integrating other modules (incl. additional exporter functions)
  - Completing consistency and integrity testing routines
  - Completing authoring environment: Extending interfaces to support specialized authoring
  - Exploring software 'props warehouses' for new scenarios:
    - Repository of objects, standard actions, desires, plans, locations, etc.
    - Everything connected to ontology

# Research contribution

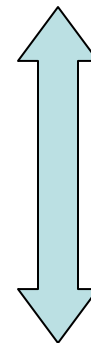
---

- **Integrated ontology-driven extensibility facilitates:**
  - Rapid authoring of new scenarios, new capabilities, new agents
  - Reduction of error, making expert knowledge explicit, etc. --- Integration, consistency checking, etc.
  - Flexible planning and scripting of training alternatives

- **But...this work is not just plumbing!**

- **We want to enable non-CS people to author new scenarios...so we're investigating the optimal point in the tradeoff between**

- Programming for Non-Experts:
  - Reduced, simple, scripting languages
  - BUT limited in functionality
  - Ex.: Sgt Blackwell internal details
- Powertools for Experts:
  - High functionality
  - BUT need considerable expertise
  - Ex.: VHUMAN's TCL, Soar, and other components



Simpler, but limited

More powerful, but complex

---

**Thank you**

