### Object - User - State Interaction

1. User touches object
2. Script 'HTTP Get's available function+animation+other-info sets
3. Script loads up a rudimentary, but functional, llDialog user interface with available action buttons.
4. Play any necessary animations
5. Is any extra information needed for the action?
   1. YES? Load interface PHP which has dynamically generated page based on action.
   2. NO? Behind the scenes, make a HTTP Request with chosen action to generic PHP interface which updates the state of the given object
6. PHP Page calculates enacted change of state and sends update to current world state database.

#### Interface elements:

*world\_knowledge\_base*: holds all *initial* and *static* information about assets and available user functions. (associated with the horizon\_three\_project database) . Information includes functions, animations, sounds, web urls (if custom interfaces are required), states, state variables etc.

*world\_state*: holds all current states of objects in the scene. When an object's state is updated because of user action, the state is updated in the *world\_state* database. This is populated when the virtual world is loaded. Initial states are taken from the world\_knowledge\_base, and immediately updated with any positional states that exist in the virtual world. It is cleared when the simulation ends. **Whether the simulation forces default positions ofobjects on load is a matter that needs to be discussed with Ross and Rune. Similarly with the clearing of the database.**

**(working draft)***PHP Scripts needed:*

*Apply result of object action as a state change. Requires database query.*

*Movement state change "At"*

*Basic object to action to database PHP state changes*

*Advanced custom object to action to input to database PHP state changes*

### Human-Goal-Workitem Interaction

A *work enactor* powers a Human User's tasks.

1. Work item is added
2. Goals are decomposed from the workitem. Goals are a list of goal states that must be satisfied to complete the goal. (Usually, there will be only one Goal for each workitem)
3. A listener is added to each goal's 'Completed' event.
4. Add goals to list of goals (organised per workitem).
5. Goals are registered with the World State Service (which can access world state).
6. Notify Human user of workitem and associated goals (labelled as sub-tasks)
7. <<World Updates>>
8. When a goal is complete, the listener is removed, and the goal is removed from the workitem's list of goals.
9. If there are no goals left on the workitem, it is checked back in as completed.

#### Interface Elements

*World State Service:* The World State Service manages various sources of state information the world is providing. It also holds a list of Goals (list of goal states) that are being tested for completeness. When a state change is detected, via polling, or events (based on the particular source type), it checks if any of the registered goals are complete.

*Human Bot:* A class to represent functions that can be performed via a human bot. These functions should include at least a notify user function so immediate information can be presented to the user.

*Work Enactor*: A work enactor is an interface that separates the work completion (via Bots) away from work assignment (via YAWL). At its core, a work enactor is given and completes work items.