reactive context-adaptive automation using heterogeneous data sources on the Web

atomsmasher is a framework which allows users to script automatically-triggered reactive behaviors by combining information from multiple, heterogeneous, time-based and static sources of data on the Web. These behaviors can be made robust by supporting multiple redundant sources for each type of knowledge, and context-adaptive by expressing behaviors in terms of simple conditions on dynamic state derived from web-based temporal data (i.e., from ATOM/RSS).

atomsmasher works by facilitating the construction of a single, consolidated world model consisting of an entity database and state model from heterogeneous web data sources. This representation is used to drive a rule-based behavior engine built in Javascript that manages efficient evaluation of rule triggers (antecedents) and firings (actions) given the sequential arrival of new information. Behaviors are themselves specified in Javascript as simple conditions and actions over objects in the world model; comparisons operators are provided that retain as similar semantics to standard Javascript operators to make it easy for web developers to transfer existing experience in Javascript towards writing behaviors.

**how it works**

1. **web data sources** - supply static data about entities (via WS APIs) or dynamic data about the state of the world (via RSS/ATOM feeds)
2. **atomizers** - unpack source-specific data structures into basic entities of world model types, e.g., people, places, resources, events/activities, and observations. Atomizers can be published to a common web repository and shared among all atomsmasher users.
3. **world model construction rules** - personalize entities generated by atomizers by merging separate entities representing the same person, place or thing, and assigning personal values. For example, a rule might merge each representation of a “person” entity generated by a social network atomizer with an entity if any) already existent in the model with a matching e-mail address. Similarly, a rule might assign the term “home” to different representations of one’s home—e.g., expressed via latitude/longitude, WiFi signal strength, or street address.
4. **state model rules** - trigger off of new items being generated by the construction rules and atomizers to drive the state model, described next.
5. **entities DB** - a repository for world model entities populated by the aforementioned rules, which are accessed from behaviors through query variables. Each query variable represents a standing query set over the entities in the DB of a particular type, which can be filtered or manipulated to derive new query variables.

**examples**

The following examples of behavior rules rely on one or more web sources each. Since behavior rules are written conditional to only world model objects, they are agnostic to the actual web sources used (shown here only for illustration).

Behaviors consist of short Javascript programs containing the following:
- query variables (sets of entities), state variables, and action (method) calls
- rules/policies
- atomizer
- world model
- entities DB
- behavior rules

1. Update my facebook status whenever I go somewhere:
   ```javascript
   if (my.location) { // "my" is the state model
     setFBStatus("At " + my.location.name, true);
   }
   ```

2. Set my away state and RSS reader mode when in meetings
   ```javascript
   if (my.activity.type.is(Activities.Meeting)) {
     // set my IM availability to away
     setIMAvailability("Away: At " + "my.activity.name + " meeting");
     // filter my dashboard RSS filter
     // for only stories tagged with this activity
     setDashFilter(function() {
       return x.tags.intersect("my.activity.tags").size() > 0;
     });
   }
   ```

3. Remind me when I have a far away appointment in advance
   ```javascript
   var max_speed_mps = .8; // walking in meters per second
   var dist = calEvent.location.dist(my.location);
   var time_needed = (dist.meters() / max_speed_mps).seconds();
   var timeuntil = calEvent.startTime - new Date();
   var extratime = 5.mins();

   if (timeuntil > 0 && timeuntil < time_needed + extratime) {
     showReminder("Excuse me sir, your " + calEvent.type.name + " is at " + "my.location.name + " meeting");
     setFBStatus("Away: At " + "my.location.name + " meeting");
   }
   ```

4. When I text myself “goodshows”, find concerts featuring artists that I have listened to recently, and friends who are free tonight that have listened to them recently too.
   ```javascript
   if (new.types.Message.Text & new.contents.contains("goodshows")) {
     // retrieve all events
     var cs = audioscrobbler.recentPlaylist();
     var playedmusic = audioscrobbler.recentPlaylist(({type:"concert", start:Now.day()}));
     var freefriends = friends().filter(function(friend) {
       return friend.cal.events.length > 0;
     });
     var goodshows = cs.filter(function(e) {
       return e.location && e.location.miles() < 22 &&
       playedmusic.artist.eq(e.artist);
     });
     reply("New, freefriends.musicPlaylist.artist.eq(goodshows.artist)");
   }
   ```