MSI Primer

http://msi.sf.net

Aron Rubin <arubin@atl.lmco.com>

Carl Hein <chein@atl.lmco.com>

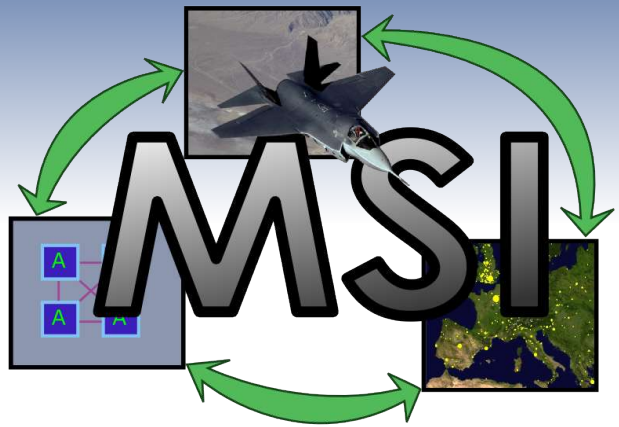
Overview

- MSI Features
- Uses for MSI
- Getting Started
- Updates vs Messages
- Wormholes
- Mirroring SoS Structure
- Optimizing MSI Communications
- Taking Advantage of Parallelism in DES
- Synchronization Points
- Upcoming Features

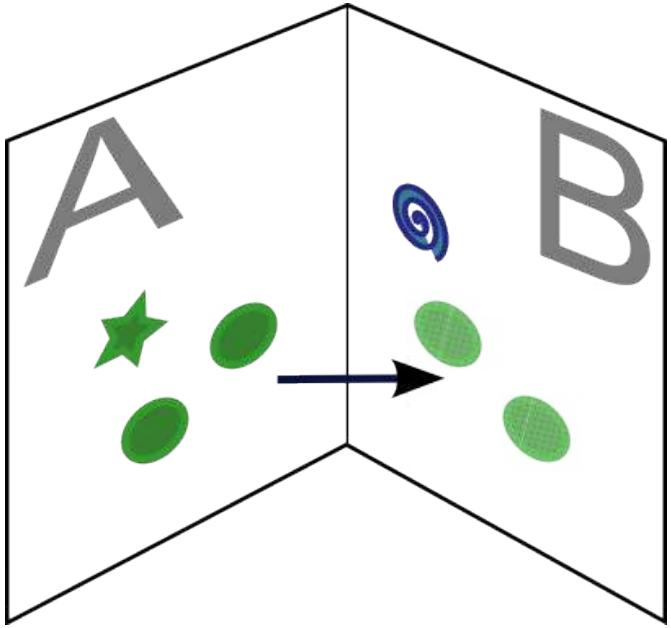
MSI Features

- Simple, direct interface in XML
- Portable, open architecture and open source
- Stable core communications API
- Fast execution. Caching and hashing
- Extensible through encapsulation and plugins

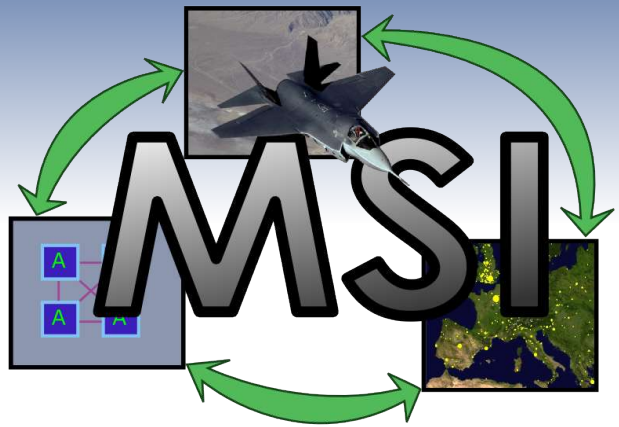
Uses for MSI
State Reflection



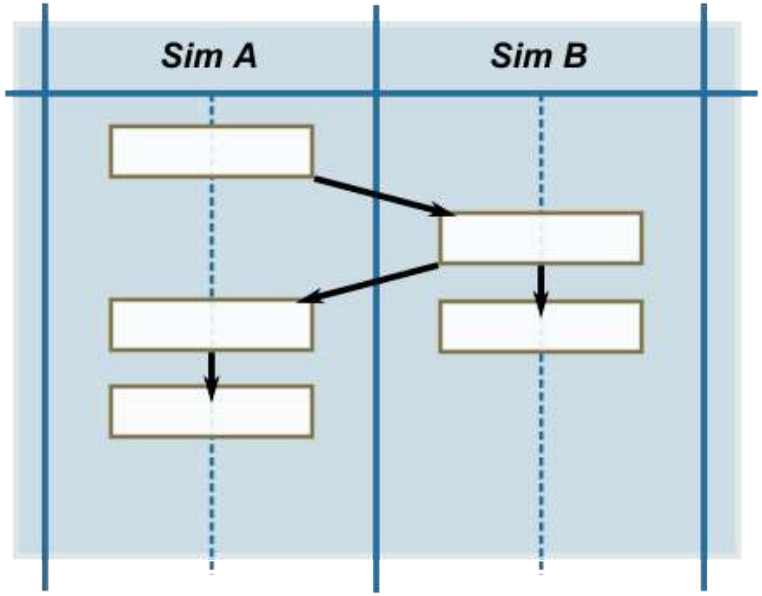
- Whenever state data needs to be reflected between two or more applications



Uses for MSI
Synchronization



- Whenever causality or order needs to be maintained between two or more applications



Getting Started
Getting Connected

- Start the MSI server
  - Usage: msi [OPTION]…
    - -d, --debug=LEVEL Set the amount of debugging prints
    - -f, --file=FILE Use FILE as a federate
    - -h, --help Print this message to the screen
    - -o, --dump=FILE Dump the transmissions to FILE
    - -p, --port=PORT Set the listening port to PORT
  - Typical Command
    - `msi -p 30001`
- Connect the client
  - `msi_client_init( use_socket, time_lock, name, port )`
  - Or snap down an MSI CSIM model

Getting Started
Publishing Data

- Decide the name and type name of your data
- Serialize and pack the data
  - Through a string representation i.e. “3.14”
  - Through base64 encoding
  - Through a complex XML representation
- Publish/Register the data
  - Using the supplied client library
    - Call `msi_send_register( name, type name, attribs, values )`
  - Through a direct socket (advanced users only)

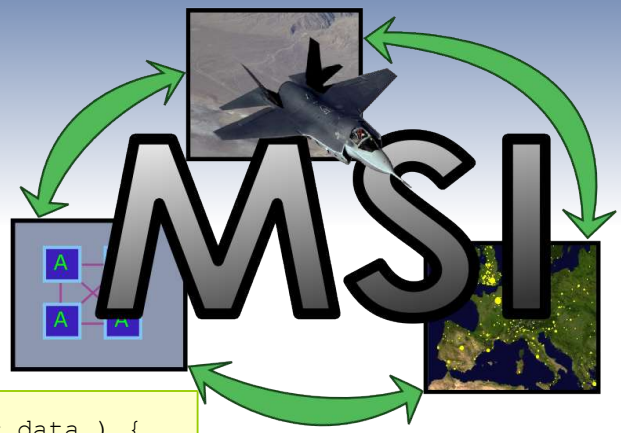
Getting Started
Subscribing to Data

- Setup an update and message handlers
  - These are functions that will be called when updates and messages come from the MSI, respectively
  - `msi_set_update_handler( handler, user_data )`
  - `msi_set_message_handler( handler, user_data )`
- Subscribe to the data by name or type and attributes
  - Call `msi_send_subscribe( name, type name, attribs )`
  - <subscribe name="" type="" attributes="" />

Getting Started
Updating Data

- IDs and names
  - MSI prefers integer IDs but will work with names
  - If a name is used the MSI will reply with both name and ID, but when only an ID is used the MSI will reply with only ID
- Update the data
  - Using the supplied client library with ID
    - If you have not stored the id call `msi_get_id( name )` New
    - `msi_send_update( id, type_name, attribs, values )`
  - Using the supplied client library with a name
    - `msi_send_update_named( name, type_name, attribs, values )`
  - <update id="" name="" attrib0="" attrib1="" attribN=""/>

Getting Started
Receiving Updates



- Making an update handler

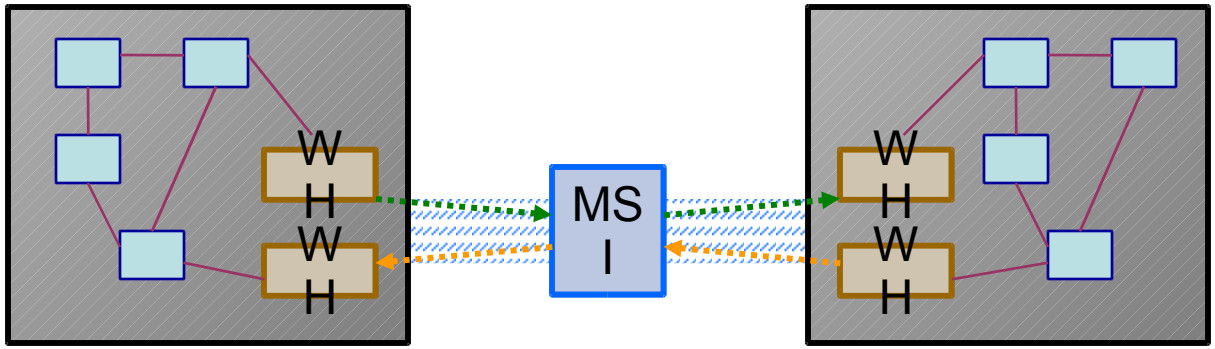
```
void my_update_handler( char **names, char **values, void *user_data ) {
  MyObject *object = NULL;
  char *name = NULL;
  long id = -1;
  int i;

  // read attributes to figure out what object we are referring to
  for( i = 0; names[i] && !object; i++ ) {
    if( strcmp( names[i], "name" ) == 0 ) {
      name = values[i];
      object = my_object_by_name( name );
    } else if( strcmp( names[i], "id" ) == 0 ) {
      id = atol( values[i] );
      object = my_object_by_id( id );
    }
  }
  // make storage for object if it did not exist
  if( !object )
    object = my_object_new( name, id );

  // loop over attributes
  for( i = 0; names[i]; i++ ) {
    if( strcmp( names[i], "attrib0" ) == 0 )
      object->attrib0 = atoi( value );
    else if( strcmp( names[i], "attribN" ) == 0 )
      object->attribN = base64_decode( values[i] );
  }
}
```

Updates vs Messages

- Updates
  - Mirror notion of object with state
  - MSI caches the values by default
- Messages
  - One time transmission of data
  - If the message is sent and a client is not there to hear it, it did not happen (for that client).
- Both
  - Have a type and name
  - Follow publish/subscribe rules

Wormhole

- Concept – messages go into box in one CSIM MSI client and comes out a box in another CSIM MSI client with the same name



- Datatypes and structure become an issue which make this approach specific to a CSIM model library
- Templates available to be tailored to a model library

Mirroring SoS Structure
Concept

- MSI can be aware of hierarchy if specified.
- Hierarchy specified through the use of the “parent” attribute
- Setting ‘A’ as the parent of ‘B’ means
  - An MSI client subscribed to ‘B’ will receive updates for ‘B’
  - An MSI client subscribed to ‘A’ will receive updates for ‘A’ and ‘B’

Mirroring SoS Structure
Distributed Stryker Vehicle Example

- Publishes
  - Client 1 publishes entity state subsystem, “stryker0.state” child of “stryker0”
  - Client 2 publishes communication subsystem, “stryker0.comms” child of “stryker0”
  - Client 2 publishes messaging subsystem, “stryker0.comms.traffic”
  - Client 3 publishes radio subsystem, “stryker0.comms.radio” child of “stryker0.comms”
  - Client 4 publishes weapons/fire subsystem, “stryker0.fire child of “stryker0”
- Subscribes
  - Client 1 subscribes to stryker0.fire
  - Client 2 subscribes to stryker0
  - Client 3 subscribes to stryker0.comms
  - Client 4 subscribes to stryker0.state, stryker0.comms.traffic
- Updates
  - Client 1 receives… stryker0.fire
  - Client 2 receives… stryker0.state, stryker0.comms.radio, stryker0.fire
  - Client 3 receives… stryker0.comms, stryker0.comms.traffic
  - Client 4 receives… stryker0.state, stryker0.comms.traffic

Optimizing MSI Comms
Through the Use of Specificity

- Subscribe to named objects instead of types where appropriate
- Subscribe restrictively first and then more and more broadly as needed
- Dynamic Context Culling
  - Subscribe to specific attributes to determine context
  - Calculate distance to context
  - If close then subscribe to other attributes
  - Example tracking visibility with object ‘C’ at x=0 y=0 z=0
    - Send <subscribe type=“X” attributes=“x,y,z”>
    - Receive <update id=“0” name=“A” x=“9.1” y=“0” z=“31.4”/>
    - Receive <update id=“1” name=“B” x=“2” y=“0” z=“1”/>
    - Calculated |C-A| = 32.7 |C-B| = 2.23
    - |C-B| < 5 so send <subscribe name=“B” attributes=“shape,color”/>

Taking Advantage of Parallelism in DES

- In most applications, one must be cautious when moving to a parallel design
  - Data hazards (read-before-write, write-before-read, …)
  - Synchronization hazards (dead-lock, live-lock, …)
- DES applications add another complication, the maintenance of causality.
  - Events must occur in the order specified
  - CSIM and MSI remove some of the data and state hazards
- Event stacking
  - Events occuring at the same sim time may execute in parallel
  - For more parallelism encourage events to occur at the same time
  - Adjust the sim time resolution reporting to MSI, course => parallel

Synchronization Points

- Not explicit in MSI
- More convenient support upcomming
- Dependencies and spin locks can be used instead
  - A MSI client could refuse to advance it's sim time until a certain dependent object is published, thereby creating a spin lock to match any dependencies it may need to wait for
  - In Stryker vehicle example client 3 may spin until stryker0.comms is published.
  - Generic Sceme
    - Clients subscribe to syncpoint0
    - Clients publish syncpoint0.client_name reached="0"
    - Clients update syncpoint0.client_name reached="1"
    - Clients wait until all syncpoint0 children have reached="1"

Upcomming Features

- Implicit federation object when session is named to ease startup synchronization
- -w, --wait_for=NUMBER|NAMES wait for clients to connect
- Scheduled messages
- Convenience functions for the client
- Web browser based control panel
- Http 1.1 based communications (for the sake of honoring the request)
- Peer-to-peer communications option