Building a Backbone for Multi-Agent Intelligent Tutoring Systems *(Work In Progress)*

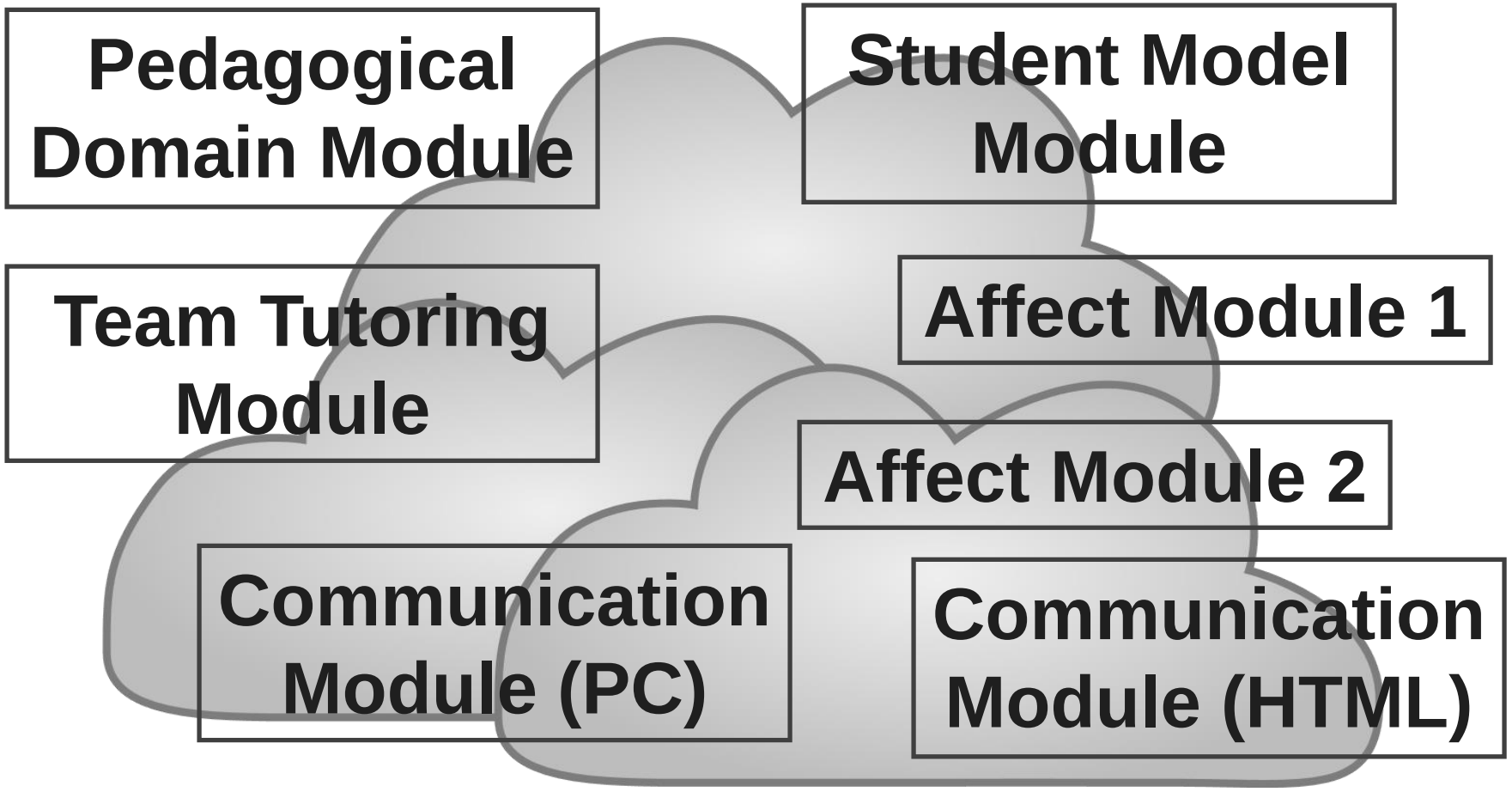
Benjamin D. Nye, David Auerbach,
Tirth R. Mehta, Arno Hartholt,

Institute for Creative Technologies
University of Southern California

May 10, 2017

The work depicted here was sponsored by the U.S. Army Research Laboratory (ARL) under contract number W911NF-14-D-0005. Statements and opinions expressed and content included do not necessarily reflect the position or the policy of the Government,
and no official endorsement should be inferred.

Distributed AIED: The Future



What is a Multi-Agent ITS?

- Goal-Oriented Services as Agents
  - Designed with a certain role/purpose
  - Adapts to available information
  - Uses best available information (avoid hard fails on bad info)

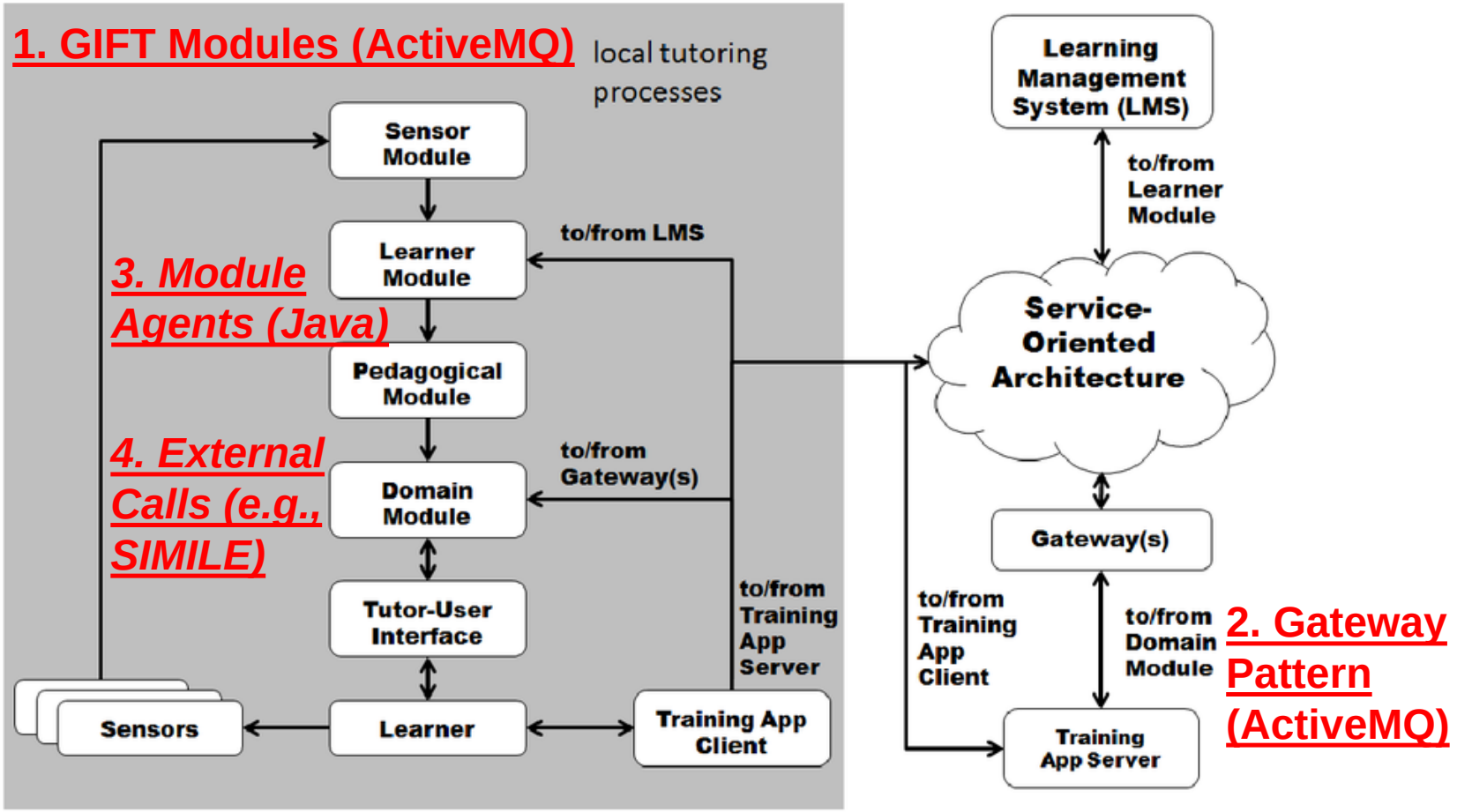
- Communication Language
  - Exchanges information using messages
  - Declarative (claims) not imperative (commands)

Goals for Multi-Agent ITS Architecture in GIFT

- Rapid Development
  - Quickly build new agents
  - Map ontologies
  - Integrate with new systems & protocols rapidly
  - Hot-swap agents

- Persistent & Incremental Learning
  - Record data about events
  - Optimize performance based on data
  - Combine data with human expert claims

GIFT Current Service Communication Patterns



Source: Sottilare, R., & Goldberg, B. (2012). Designing adaptive computer-based tutoring systems to accelerate learning and facilitate retention. *Cognitive Technology*, *17*(1), 19-33.

Additions for GIFT Targeted in this Phase

- Registering Services: Adding Services at Runtime
  - Currently: Only possible using ActiveMQ protocol
  - Need to have course declare mandatory/optional services
  - Need to resolve/hotswap between alternate services

- Interoperability: Frameworks with Different Messages
  - Currently: Only supports GIFT registered messages
  - Need translate between different message ontologies

- Unified Agent Pattern
  - Currently: Agents only experimentally hard-coded in modules
  - Need a general way to attach agents to modules
  - Need a place for agents/gateways that are not module-specific

The Target Process: Registering Agents at Runtime

1. Module & Agent Design Step: Modules and agents designed to receive/send specific message types

2. Ontology Map Step: Declare mappings between messages from different frameworks in a generic serialized format

3. Course Design Step:
   - Configure course to declare module software agents (e.g., Java)
   - Configure course to listen to external gateways/repositories for agents
   - Configure to declare what ontology mappings are used by each gateway

4. Course Runtime:
   - Initialize agents in each module: Start Java gateway listeners/services
   - Agent Messaging: Send/receive. Advanced patterns (e.g., proposals) occur.

The Target Process: Registering Agents at Runtime

1. **Module & Agent Design Step:** Modules and agents designed to receive/send specific message types

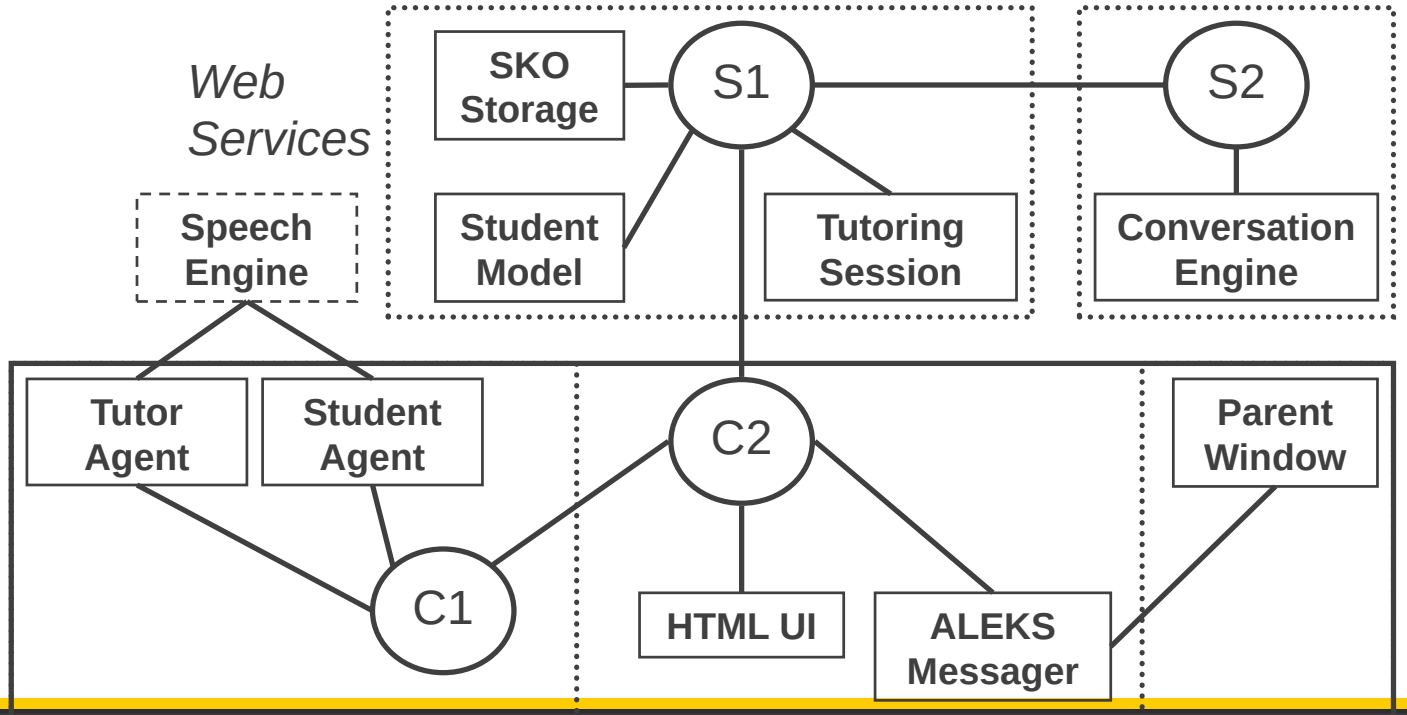
2. **Ontology Map Step:** Declare mappings between messages from different frameworks in a generic serialized format

3. Course Design Step:
   - Configure course to declare module software agents (e.g., Java)
   - Configure course to listen to external gateways/repositories for agents
   - Configure to declare what ontology mappings are used by each gateway

4. Course Runtime:
   - ***Initialize agents in each module: Start Java gateway listeners/services***
   - **Agent Messaging: Send/receive**. Advanced patterns (e.g., proposals) occur.

Registering: SuperGLU (Generalized Learning Utils)

- Service-Oriented: All functions in services
- Gateway Model: No central bus, uses distributed gateways
- Anonymous Services: Messages filtered by semantics



SuperGLU: Why was it made?

- Rapid service/agent creation:
  - Build to receive and send information (messages)
    - Emerging message ontology: Standard ITS logging
  - Abstract connection to storage that saves/queries data
  - HTML5: Client-side services in JS
  - Docker containers: Spin up on cloud server

- Hot-swapping:
  - Like a distributed publish/subscribe or fan-out
  - Plug-and-play (connect/disconnect service to a gateway)
  - No central hub (connect/disconnect gateways on-the-fly)

SuperGLU: What does it add?

- Services should care about information
  - Content of message should determine handling
  - Services should act like agents (e.g., “No reply? Ask again!”)

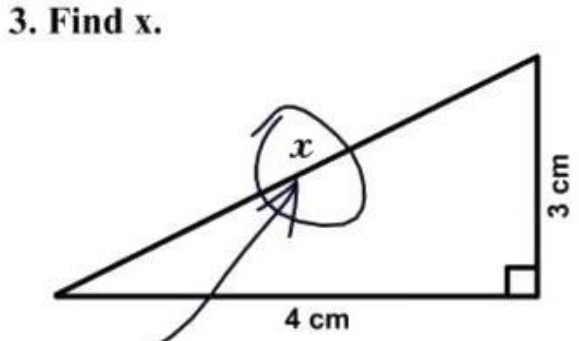
- Message Bus Wrappers:
  - Each service only sees its own gateway
  - Gateways handle message transport:
    - Cross-domain JS services (HTML5 postMessage)
    - Real-time client-server (socket.io Websockets)
    - Server-to-server (wrap protocols like ActiveMQ/AMQP/Stomp)
  - Only gateways know about URL’s and networks

- GIFT Integration Points:
  - AgentContainer Module: New module to register agents/gateways
  - Existing Modules: Register agents inside a module also

Ontology Mapping: Translating Messages

Sometimes, messages can be unclear:

System A Sent:



System B Replied: Here it is

Ontology Broker solution:

- Register & store explicit ontologies
  - Multiple Types: Message formats and types
- Convert messages from one ontology to another

Ontology Broker: High Level View

- Three ways to do ontology mapping:
  - **Declarative (e.g., from OWL/JSON): This is the one currently implemented**
  - Direct functions (e.g., custom converters): Simple to implement, but would be for ad-hoc cases
  - Remote fall-through: Send request to external OntologyBrokerService. Also simple, but would just be a dispatcher to one of the other two.
- Inventory of messages and mappings are in a Google Doc
- About a dozen mappings implemented that we are testing right now
- Broker searches for any series of maps from the current message to an endpoint in the target format

OntologyBroker

- mappings: List of <MessageMapping> to detect that a mapping can be made and do that mapping

- defaultTemplates: <list> of partially-complete messages that can be finished using a map

FindPathAndConvertMessage(BaseMessage inMsg, MessageType inMsgType, MessageType targetMsgType, boolean strict): Main function call for conversion

Note: By far the most common field type, a value in some nested lists, dicts, token/objs

FieldData
- data: obj

NestedFieldData
- indices: List of <container type, key>, so < str, int/float/str/bool>

MessageMapping
- InMsgType = MessageType>
- OutMsgType = MessageType
- InDefaultMsg = MessageTemplate
- OutDefaultMsg = MessageTemplate
- FieldMappings = List of <FieldMaps>

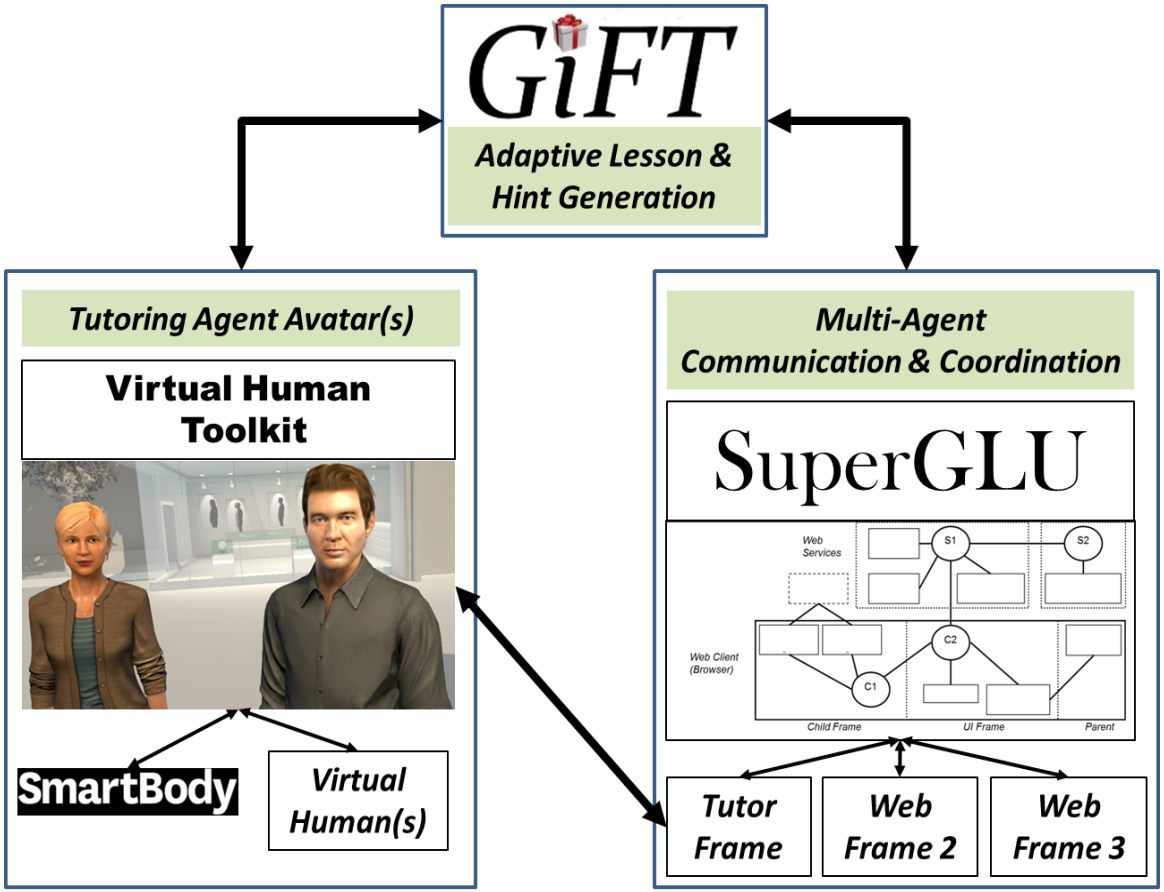
MessageType
- name = str
- minVersion = float or null
- maxVersion = float or null
- MessageTypeTemplate = MessageTemplate

MessageTemplate
- DefaultFieldData = List of <FieldData>

FieldMap
- InFields = List of <FieldData>
- OutFields = List of <FieldData>
- Conversion = <DataConverter>

Framework Case 1: Integrating OS VHT

Proof of Concept:
Unify GIFT, VH, and SuperGLU ecosystems



OSVHT Coming Soon at:
https://github.com/USC-ICT

GLU Tools at:
https://github.com/GeneralizedLearningUtilities

Virtual Human Toolkit

GIFT

Virtual Human Toolkit

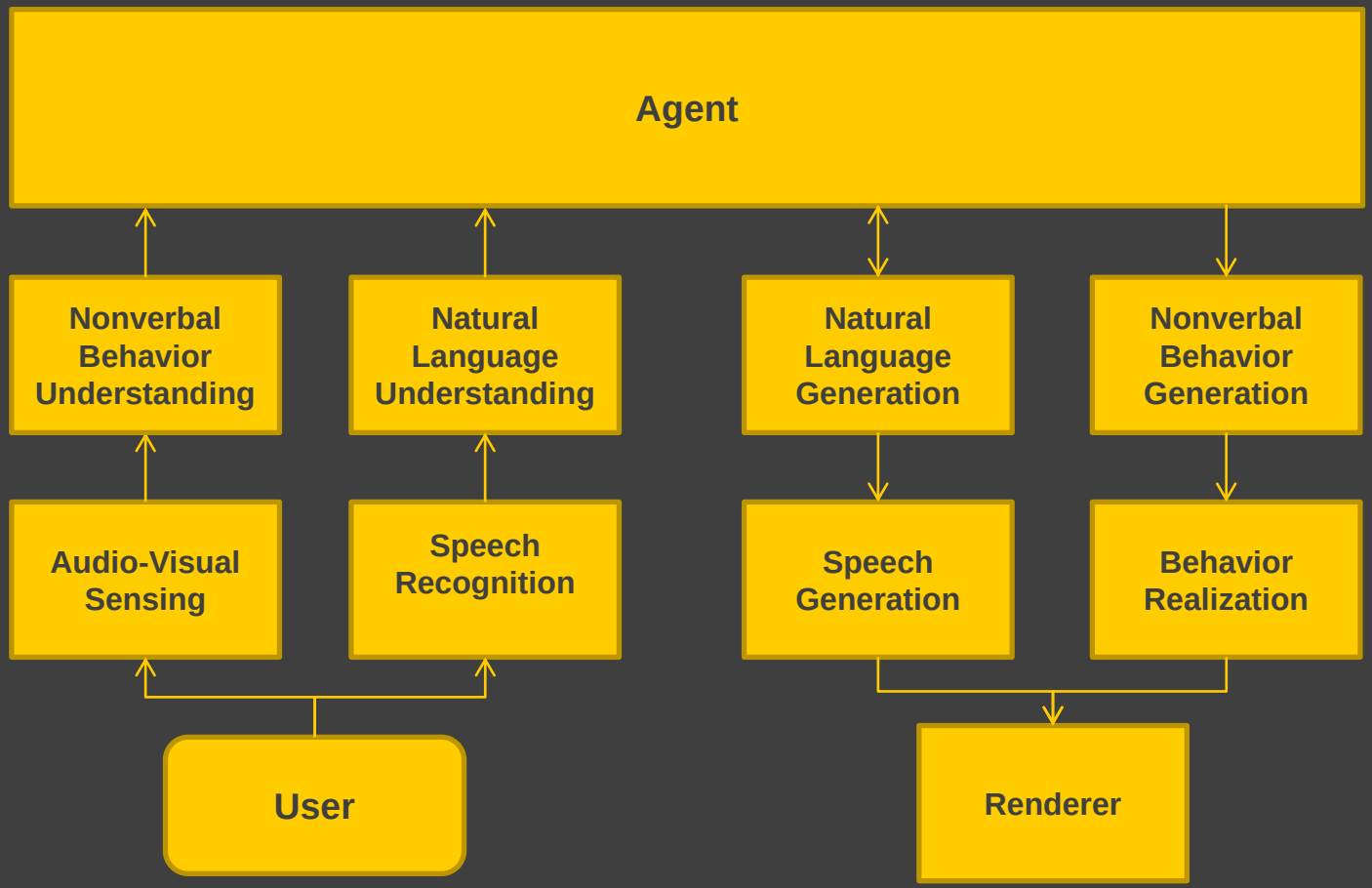
Virtual Human Toolkit

- Offers components that cover
  - Audio-visual sensing
  - Automated speech recognition
  - Natural language processing
  - Nonverbal behavior generation
  - Behavior realization
  - Text-to-speech
  - Rendering
- Integrated as part of a modular, flexible architecture
- Based on SAIBA, BML and FML standards
- Allows mixing and matching with one's own technologies
- Government Purpose Rights (GPR) version available

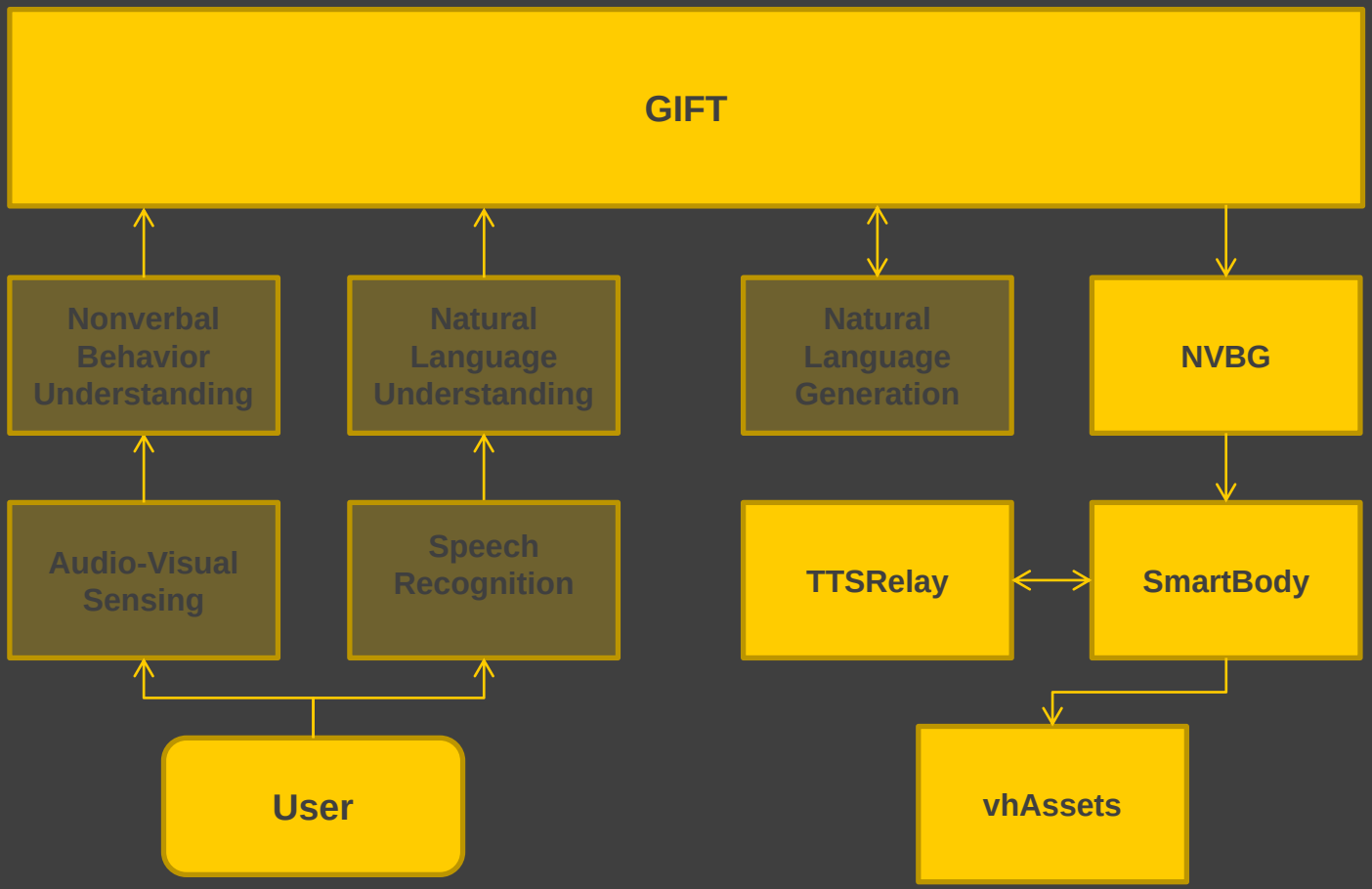
Open Source ICT Virtual Humans (OSVHT)

- Open source subset (OSVHT), intended to include:
  - NVBG (generates nonverbal behavior from surface text)
  - SmartBody (procedural animation system)
  - vhAssets (Unity game engine wrapper)
  - TTSRelay (unified interface to various TTS systems)
  - VHMsg (message protocol based on ActiveMQ)
  - Logger (logs messages)
  - Launcher (launches and manages modules)
- Three characters to choose from
  - Army Male
  - Civilian Male
  - Civilian Female

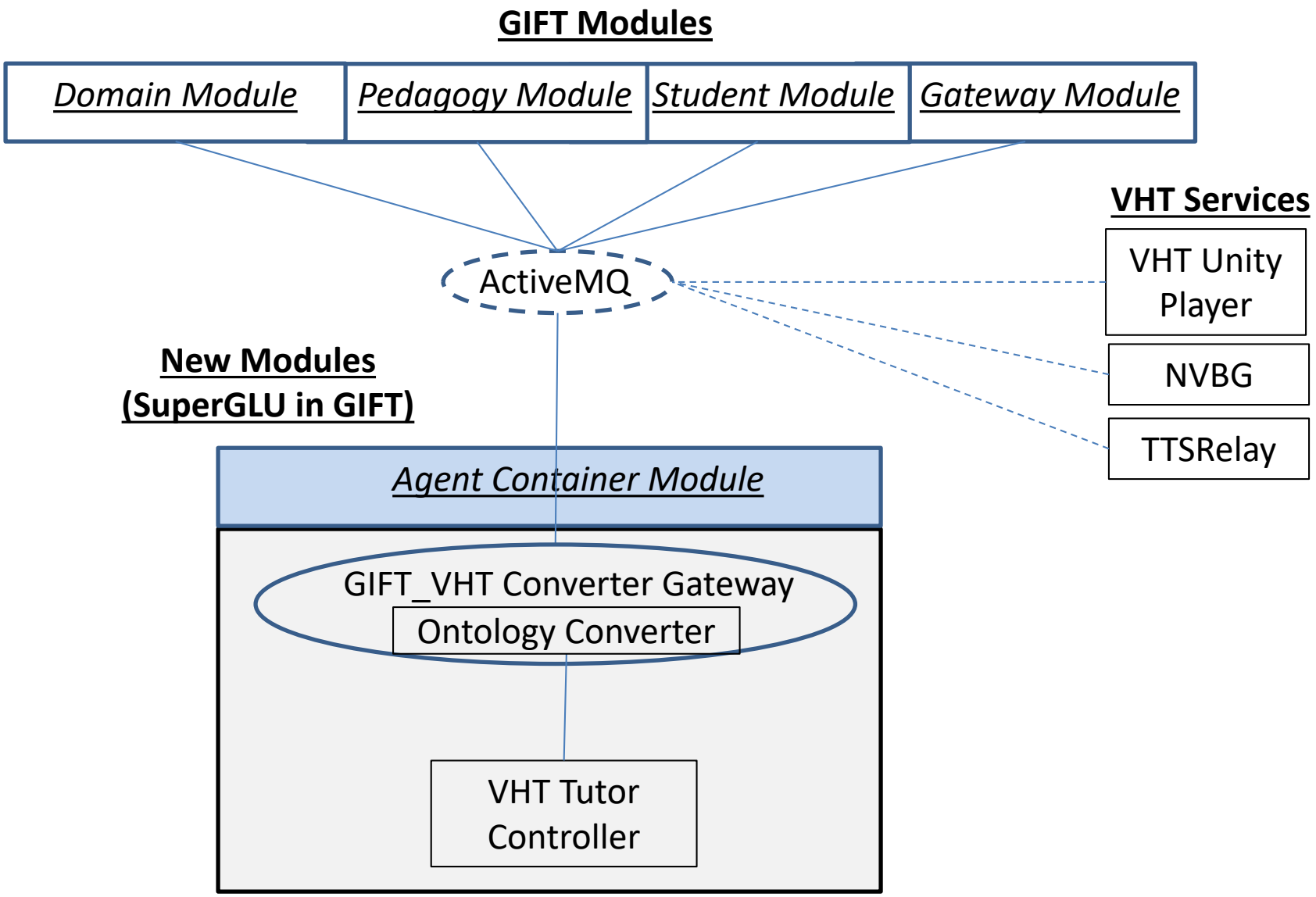
Virtual Human Architecture



OSVHT


GIFT
Nonverbal Behavior Understanding
Natural Language Understanding
Natural Language Generation
NVBG
Audio-Visual Sensing
Speech Recognition
TTSRelay
SmartBody
User
vhAssets

Implementation


GIFT Modules
Domain Module
Pedagogy Module
Student Module
Gateway Module
VHT Services
ActiveMQ
VHT Unity Player
NVBG
TTSRelay
New Modules (SuperGLU in GIFT)
Agent Container Module
GIFT_VHT Converter Gateway
Ontology Converter
VHT Tutor Controller

Live Demo

(Hold on to your hats…)

1. Agent Container Module listens to messages for all modules, relaying them to *GIFT_VHT Converter Gateway* (implemented in GIFT using a Java port of the SuperGLU library). Most messages that reach the *GIFT_VHT Converter Gateway* converter are ignored because they are have no relevant mapping in the VHT message ontology (which is its target).

2. GIFT sends a *GIFT:Display Guidance Tutor Request* from the Domain Module.

3. The *GIFT:Display Guidance Tutor Request* is received by the *GIFT_VHT Converter Gateway.*
4. The gateway's built-in ontology converter recognizes that a valid mapping chain exists for this message to create an VHT message.

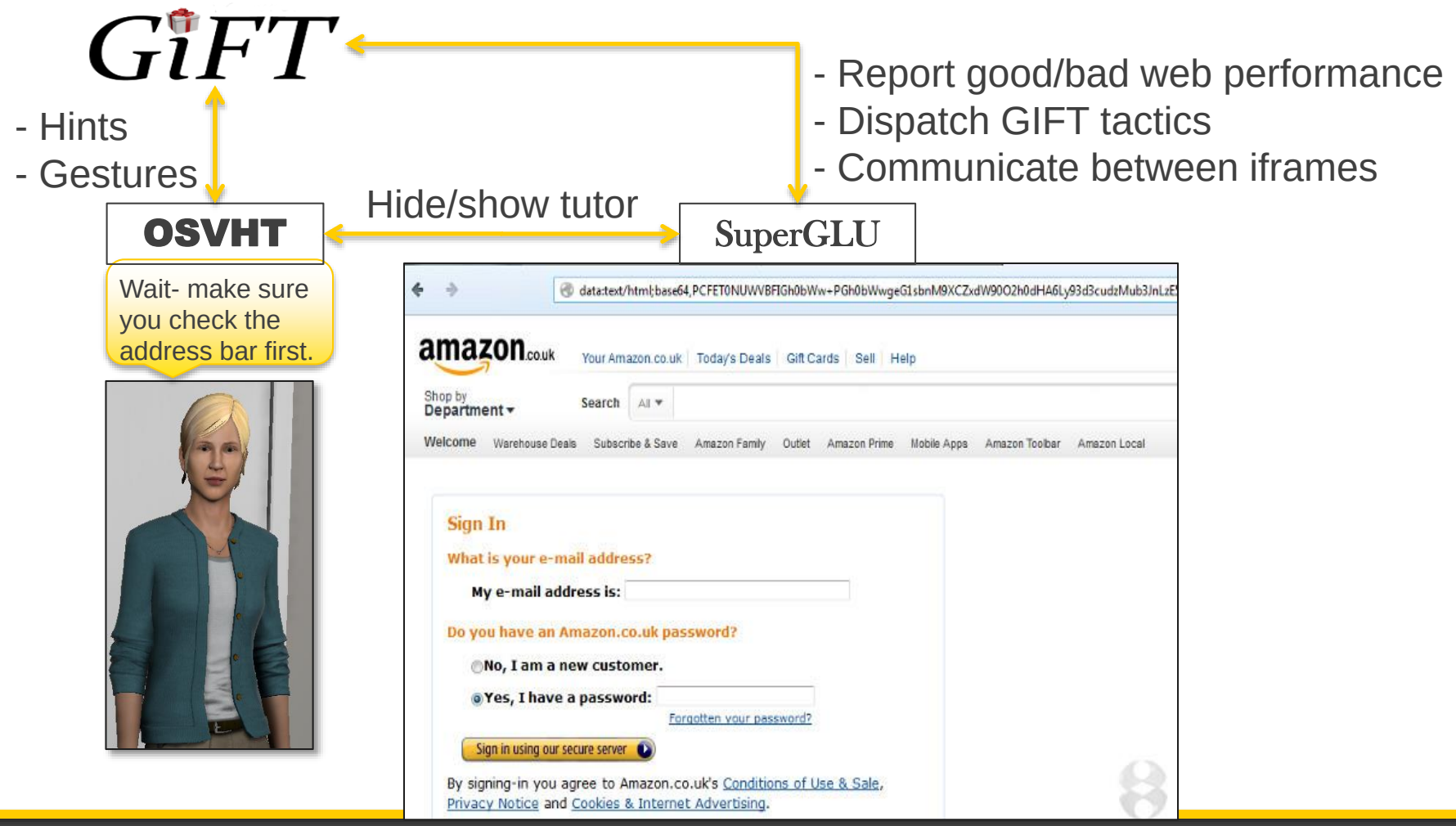
5. The ontology mapping converts the message into a *SuperGLU:Speech* message. The converter then maps the *SuperGLU:Speech* message into a *VHT:vrExpress* message.

6. The *VHT:vrExpress* message reaches the *VHT Tutor Controller*, which modifies the message to set any required optional parameters (e.g., the name of the speaking agent). The OSVHT Tutor Controller currently exists to maintain values for default parameters.

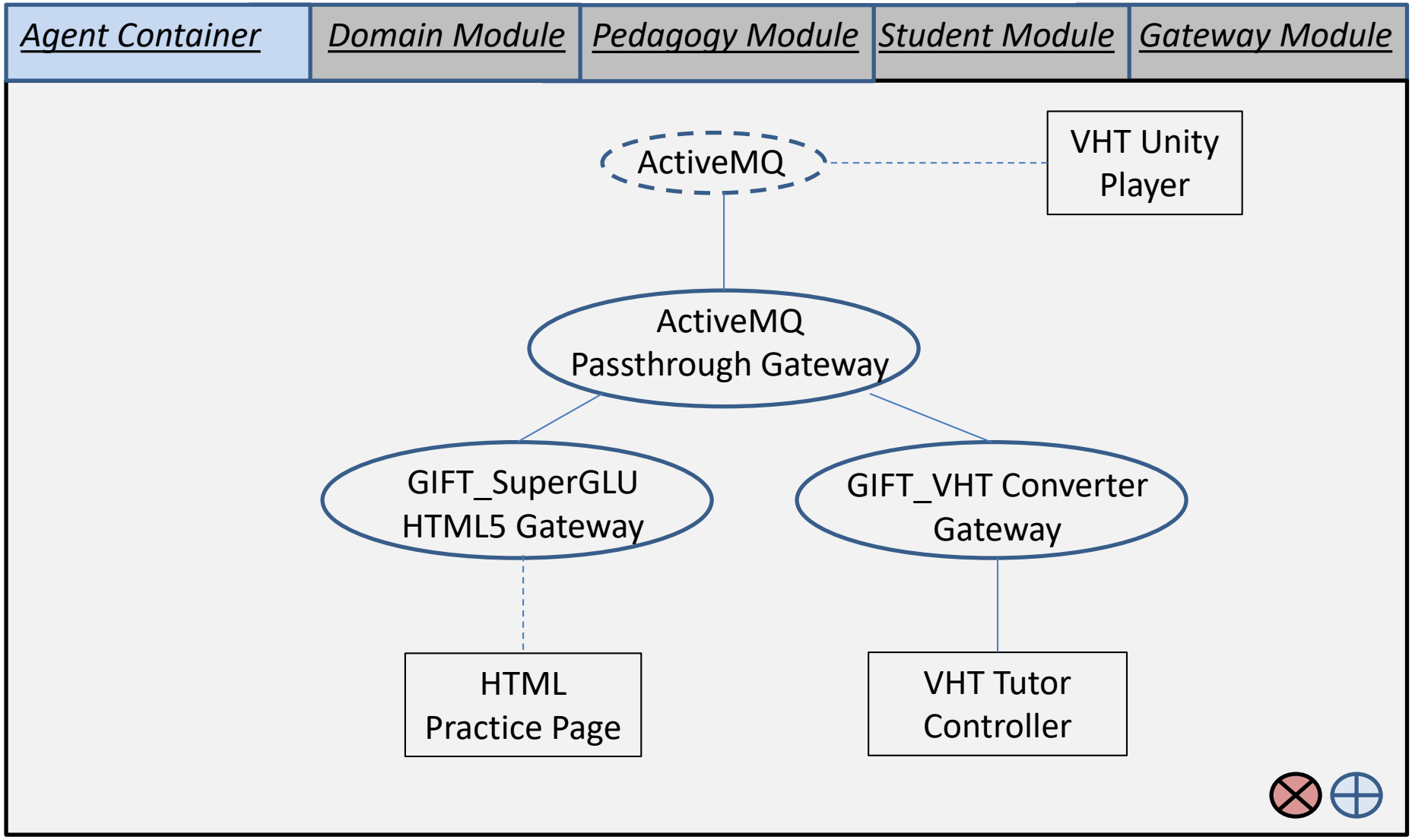
7. The *VHT Tutor Controller* sends a new *VHT:vrExpress* message back to the *GIFT_VHT Converter Gateway.* The gateway does not convert it before passing it to ActiveMQ, because relays OSVHT messages.

8. The NVBG receives the *VHT:vrExpress* message through ActiveMQ. NVBG generates non-verbal behaviors that match the words given, and both the speech and behavioral markup are sent to the OSVHT Unity Player. The tutor says something in the OSVHT Unity Player, by transmitting messages to the TTSRelay to generate speech while the accompanying animations are executed by the character in the OSVHT Unity Player.

Next Example: Lesson on Cybersecurity (Phishing)



Case 2: Viewing Agent Container Module Gateways and Services



New Capabilities Implemented

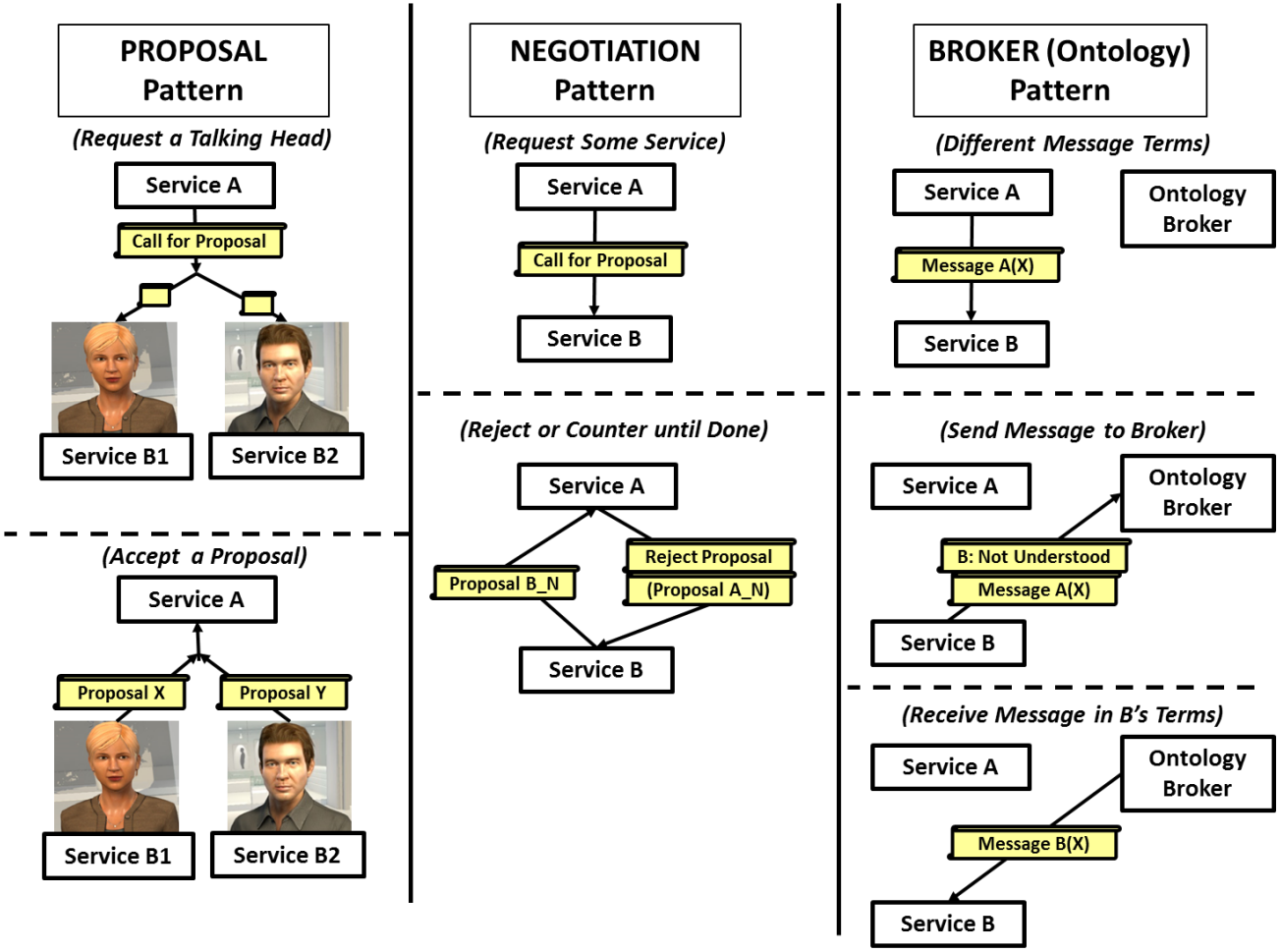
- Rapid Development
  - Quickly build new agents:
    - **SuperGLU open source for HTML5 services + Python services**
    - **Virtual Human pedagogical agent for GIFT**
  - Integrate with new systems rapidly
    - **SuperGLU HTML5: Build JS services or wrap existing**
    - **SuperGLU Cloud: Build new cloud services or wrap**
    - **Virtual Human Ecosystem: New modules built regularly**

- Next Steps (in progress):
  - Cybersecurity proof-of-concept mini-course
  - Usability study: CS students creating & adding new services

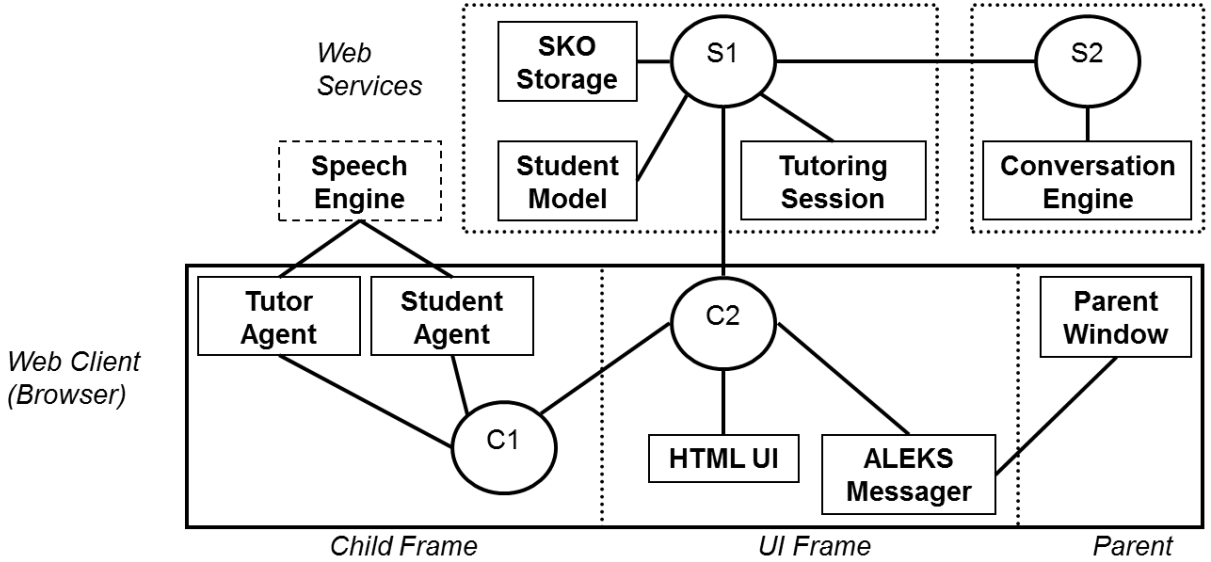
Future Work:

Advanced Multi-Agent Communication: Proposals & Brokering

Key Communication Patterns



Problem: Plug-and-Play for Distributed Agents



How to get web services to talk easily?

- Proposals: Solves many-vendors issue. Ask for services willing to give the info, and accept a reply bid (if any match).
- Negotiation: Solves coordination/search problems. Counter proposals occur until a match found or quitting.

Problem 2: Extending Ontologies & Mappings

Message Formats to Convert:

1. SuperGLU (Standard ITS Logger)
2. OSVHT Messages
3. GIFT Messages
4. *(xAPI Messages)*
5. *(DataShop/LearnSphere Messages?)*

Status: Initial mappings in place, but limited overlap between these ontologies.

Questions & Discussion