Building a Backbone for Multi-Agent Tutoring in GIFT (Work in Progress)

Benjamin D. Nye[1], Daniel Auerbach[1], Tirth R. Mehta[1], Arno Hartholt[1]
University of Southern California, Institute for Creative Technology[1]

INTRODUCTION

As intelligent tutoring systems (ITS) increasingly need to interoperate and co-exist, emerging systems have transitioned toward service-oriented designs to enable modularity and composability of tutoring components made and/or maintained by different research and development groups. However, as a research community, we have still not reached a point where it is trivial for a new service to be added into a system like the Generalized Intelligent Framework for Tutoring (GIFT; Sottilare, Goldberg, Brawner, & Holden, 2012). In an early paper considering this issue with respect to the GIFT architecture (Nye & Morrison, 2013), we proposed addressing this issue by building toward a lightweight multi-agent architecture where certain services act as autonomous agents: "*a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it senses in the future*" (Franklin & Graesser, 1997; p. 25).

In our work in progress described here, we discuss how we are approaching the opportunity to build such capabilities into GIFT. The high level goals of our work are targeting two core goals for GIFT: A) to be a lightweight framework that will expand access to and use of ITS and B) to help GIFT to increase the intelligence and effectiveness of its services based on data over time. We are currently targeting the first goal, which will underpin the second goal. However, what does it mean to be a lightweight framework? In this context, a "lightweight framework" is framed as minimizing the following criteria: (1) hardware requirements, (2) software expertise to design services, (3) software expertise to use existing services, (4) software expertise to stand up the message-passing layer between agents, and (5) a minimal working message ontology (Nye & Morrison, 2013). Since our original paper four years ago, GIFT has made significant strides in reducing barriers related to hardware by building a cloud-based version and software expertise to use GIFT services through authoring tools. It has also developed a growing ontology of messages (e.g., https://gifttutoring.org/projects/gift/wiki/Interface_Control_Document_2016-1). With that said, despite now-extensive documentation, designing new services for GIFT is still not trivial and strong expertise is required to pass messages between GIFT modules and agents (either internal or external).

To address these issues, the Building a Backbone project is working toward agent-oriented designs that build on GIFT's existing service-oriented framework. By moving from services toward agents, modules will be able to act more autonomously, enabling capabilities such as plug-and-play, hotswapping, and selecting between multiple services providing the same capabilities. These new capabilities are intended to reduce barriers to building new GIFT-compatible services and also to integrating GIFT with other service-oriented ecosystems. The first steps toward these capabilities are an ontology mapping service and an initial integration that combines GIFT, the Virtual Human Toolkit core framework for agents, and the SuperGLU framework for adding agent-oriented capabilities for coordinating services. This paper reports on work to date, with an emphasis on target capabilities, design decisions, challenges, and open research questions for this work.

TARGET CAPABILITIES FOR GIFT AS A MULTI-AGENT ITS

When considering the advantages for GIFT as a multi-agent framework, we must consider the question: advantages for who? What stakeholders benefit and how? GIFT has at least six different types of users:

- *Learners (Use Services):* Students and trainees who participate and learn from GIFT courses
- *Instructors (Use Services):* Deliver, modify, and possibly design GIFT courses
- *Basic Course Designers (Configure Services):* Modify or build course content with wizards/tutorials, including selecting or configuring a group of services verified to work together.
- *Advanced Course Designers (Compose Services):* Build advanced content and adaptive behavior, by selecting and configuring services to work together as a group
- *Service/Agent Programmers (Make/Add Services):* Code new services or agents used by GIFT
- *Framework Providers (Combine Service Ecosystems):* Maintain and interface other large frameworks with GIFT

The above list indicates a stakeholder type, their expected role for interacting with services/agents, and then a brief description their primary goal and role when dealing with GIFT. In terms of direct effects, the latter three stakeholders who build and compose the services can benefit from greater modularity, reusability, and interoperability. By lowering barriers to entry, agent-oriented design should encourage a growing ecosystem of intelligent and interactive services. A comparison can be made with modern web design, where seemingly simple HTML pages actually involve interacting JavaScript and frames hosted by dozens of web services that each specialize in particular functionality (e.g., ad hosting, static content delivery, feeds of weather information). From the standpoint of less-technical stakeholders, the benefits are indirect and would be evident by a broader and more effective set of adaptive courses. For Learners and Instructors, a successful multi-agent ITS should be "magic" in that it just works and is effective without requiring any knowledge about what agents, services, and frameworks do under the hood. For Course Designers, the standard is "plug-and-play" where advanced capabilities such as services and agents should have reasonable defaults and clear boundary conditions/suggestions for when they can be used. For Service Programmers, the goal is that new agents and services can be quickly integrated to communicate with core GIFT services. For Framework Providers, the goal is to make it straightforward to complete either a one-time or iterative integration that enables a suite of existing services (with its own messaging/communication) to interact with GIFT *and any other services or frameworks that already work with GIFT* (i.e., GIFT as a hub for interoperability).

To accomplish this, a multi-agent framework needs to satisfy certain criteria:

1. Agent Communication Language(s): Services should communicate using explicit messages, where every service has a clear set of messages that it sends and receives (i.e., a message set analogous to an asynchronous API). Services do not necessarily need to use the same message set, provided that some mapping is available to map messages between ontologies.
2. Plug-and-Play: Services must not be hard-coded to send messages to any specific other service, but instead must either send messages with no explicit recipients (i.e., blind sending) or must resolve their specific service communication at runtime (e.g., proposal patterns, negotiation).
3. Independent Agency: Every service should be designed to contain no assumptions or minimal assumptions about how other agents process information internally, and instead rely on messaging to determine declared or inferred states about other services it interacts with. This lends itself to declarative messaging, where instead of a standard command of "Start <x>" and assuming that

<x> is started, a service might instead message "Request Start <x>" and watch for a message of "Inform <y service> Start <x>."

4. Network Structure/Protocol Independent: Services should be able to plug in using a single protocol rather than requiring a separate API for every protocol (e.g., Websockets, AMQP, HTML5 postMessage), so that the content and handling of messages is decoupled from how the message is transported. Related to this, it must be assumed that some services may need to pass messages through multiple systems or protocols to reach their final destination, but should not require any knowledge of the network structure or protocols to do their job, unless their job is specifically-related to communication protocols.
5. Graceful Failure: Services should have certain goals (implicit or explicit) that they attempt to achieve, based on the best information available (i.e., messages received). In the case of a violation of expected communication (e.g., requests receive no replies, an ill-formed message is handled), each service should continue to try to achieve its goals. Errors and poorly-behaving services should be recorded to enable fixing or removing problem services.

For a shorter summary of these principles, it could be said that services for a system like GIFT should act like humans do as agents in a well-run team: we know what job and goals we are trying to do, we do not mind-read (i.e., assume interal processes of other humans) or need to know the whole organizational structure (e.g., the network), and if other people fail to give us what we need to do our job then we still do our best and/or report those people as a problem for the team. For some services, such as reactive and stateless calculations, these principles are easy to follow. For others, such as an agent in a complex simulation, it can be more complex. However, without these principles, agents become strongly coupled to either other services or to fragile factors irrelevant to their actual goals (e.g., network structure). A multi-agent architecture following these principles offers many advantages over traditional designs:

- ***Specialization:*** In a multi-agent system, agents can specialize in tasks. While this benefit also applies to other component-based systems, agents can communicate their abilities using semantic messages and agent communication languages (Bellifemine et al., 2008).
- ***Decentralization:*** Modern software environments rely on a large number of interacting components, where even a single web page is actually an orchestration of many web services, domains, platforms (e.g., desktop vs. mobile), and a mixture of various client and server-side code. ITS need to follow these patterns, through service-oriented and agent-oriented designs.
- ***Customization:*** An implication of distinct agents that communicate using messages is that each agent does not rely on the specific internal state of any other agent. This means that each agent can be rapidly customized, provided that it still uses the expected messages.
- ***Data-Driven Enhancements:*** Related to customization, this means that it is also possible for agents (and simpler services) to use collected data to improve their performance against certain metrics and goals.

With this said, GIFT is not a project starting from the ground up: it is an existing, substantial framework that already follows certain software architecture patterns and has certain assumptions. To add these capabilities while retaining its existing strengths is an ongoing research project, which will be described in the following section.

HIGH LEVEL DESIGN

In this first year of the project, the stakeholder use-cases that we are focusing on are *Framework Providers* and *Service Programmers*. To build toward the case of integrating new framework providers into

GIFT, we are building a branch of GIFT that integrates services from an upcoming open source release of Virtual Human Toolkit (Hartholt et al., 2013) by using the SuperGLU (Generalized Learning Utilities) framework (Nye et al., 2014). These three frameworks each have unique strengths and architectural principles. Of these, SuperGLU's role is to provide fundamental framework components to abstract away issues related to plug-and-play capabilities, network structure independence, translating between agent communication languages, and service agency. By comparison, GIFT and the Virtual Human Toolkit are both longer-standing frameworks with substantial existing services and content are being connected as an example of integrating a framework provider (i.e., the Virtual Human Toolkit). The complementary capabilities of these frameworks are that:

GIFT (Generalized Intelligent Framework for Tutoring; Sottilare et al., 2012) is an open-source ITS architecture that implements a service-oriented architecture for tutoring, which can be deployed as a standalone installation, a network client/server architecture, and a cloud-based framework. Through its functionality for authoring tools and ITS course package managers, GIFT offers both fine-grained capabilities (e.g., services for student modeling) and larger-grained tools.

The **Virtual Human Toolkit** is a set of interacting services that primarily focus on training systems where highly-realistic animated agents act as mentors or act out interactive scenarios as virtual roleplay-ers. The Virtual Human Toolkit contains a number of services, including animated agents (typically rendered in Unity), the nonverbal behavioral generator (NVBG) to automatically determine gestures based on speech, and SmartBody to coordinate agent behavior (e.g., locomotion, lip syncing). In addition to its primary services, Virtual Human Toolkit also provides message patterns that help for registering and controlling services, leading it to be used as a messaging framework to coordinate behavior of other services in ITS, such as goal-oriented agents, natural language processing and audio-visual sensor streams (Kenny et al., 2007). The Virtual Human Toolkit is based on the SAIBA framework (Vilhjálmsson et al., 2008) and utilizes several messaging standards (Kopp et al., 2006; Heylen et al., 2008; Scherer et al., 2012). It underpins a number of ITS and training systems (Campbell et al., 2011; Kenny et al., 2007).

The **SuperGLU** framework is an ongoing open source project designed for real-time coordination of different ITS services, based on multi-agent communication. SuperGLU branched out of the Memphis team for the Office of Naval Research (ONR) STEM Grand Challenge, where it was implemented as the Shareable Knowledge Objects ITS (SKO-ITS) architecture (Nye et al., 2014). This system uses explicit semantic message-passing, where messaging is based on two standards: the ADL xAPI standard for reporting learning experiences (Murray & Silvers, 2013) and the FIPA agent-communication language standard (FIPA, 2013). This framework uses special "gateway" services to abstract away the device and network architecture (e.g., client/server; cross-domain messaging). This enables each individual service to focus entirely on the messages that it sends and receives, rather than being programmed with knowledge about the specific services generate the messages. This system is currently being used for the ONR ElectronixTutor project, where it is being used to integrate four separately-developed ITS from Worches-ter Polytechnic Institute, the University of Memphis, Raytheon-BBN, and Arizona State University (github.com/GeneralizedLearningUtilities/SuperGLU). A pre-prototype version of the SuperGLU architecture supported the NewtonianTalk project, which integrated AutoTutor for dialog-based tutoring, Physics Playground for simulation-based learning, and GIFT (Ventura et al., 2015). SuperGLU currently has implementations of its core functionality in Python, JavaScript, and Java.

These three frameworks offer distinct advantages that will be leveraged when combined into a multi-agent framework. GIFT, with its high-level capabilities for course management and pedagogical strate-gies, offers an integrator architecture that can leverage new services that plug into the system. It also provides significant infrastructure for tutoring simulation-based training.offers a bridge to a larger ecosystem of existing services and tools (e.g., high-fidelity tutoring agents and avatars) which will continue to expand as new applications use and build upon an open source release of the Virtual Human

Toolkit (OSVHT). Finally, the SuperGLU framework will support rapid creation and integration of new services, with a particular emphasis on web-based ITS, cloud-based services, and multi-agent systems. Integrating the SuperGLU framework and its expanding set of services with GIFT will provide a variety of specialized services and tools that accomplish common ITS tasks, ranging from cloud-based storage (e.g., the GLUDB project; github.com/memphis-iis/GLUDB) to service wrappers for HTML animated agents (Nye et al., 2014). Based on this integration, our work consists of four phases of functionality: 1) Framework Interoperability (combining existing services and ontologies), 2) Service Design (creating and adding a new service), and 3) Service Composition (composing multiple services to work together), and 4) Advanced Agent Capabilities.

Framework Interoperability (Functionality to Combine Frameworks/Services)

The current work on this project is building basic Framework Interoperability capabilities, focusing on the use-case using the GIFT, SuperGLU, and the Virtual Human Toolkit these frameworks together. This phase will culminate in a proof-of-concept reference implementation where a GIFT course interacts with an OSVHT animated pedagogical agent who reacts to behavior on multiple websites that communicate with GIFT in real-time using the SuperGLU framework (depicted in Figure 1). A proof-of-concept GIFT mini-course will demonstrate (visually and programmatically) the functionality and capabilities of this integration, along with any helper services to streamline this process for future GIFT course designers. We plan to have this mini-course to present the topic of phishing, through interacting frames that attempt to trick the user into providing information through insecure methods (e.g., http vs. https) or to the incorrect site. This will be facilitated by using SuperGLU to coordinate communication between iframes on multiple domains.

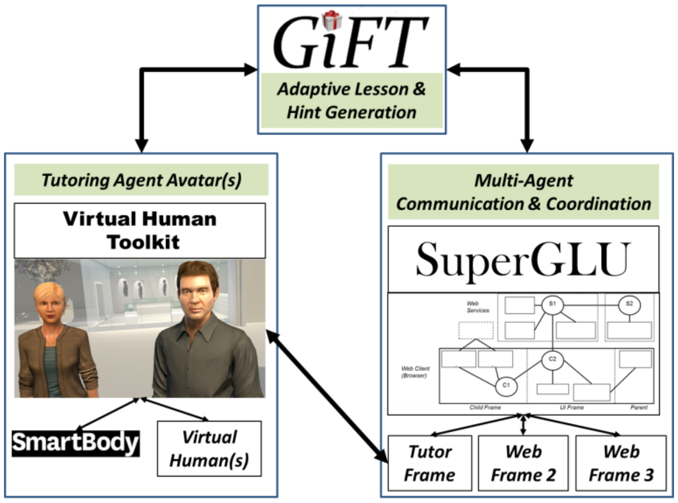

Figure 1: Concept of First Integration Use-Case

Service Design (Making and Adding New Services)

Based on the basic capabilities put in place for the first phrase of research, we will create user-facing tutorials and documentation about how to perform certain common multi-agent ITS behaviors, such as building a Virtual Human tutor, creating a new cloud-based SuperGLU service, or using SuperGLU to integrate cross-domain services. To improve and evaluate the usability of this from the service design standpoint, we will recruit approximately novices (e.g., students) to complete these tutorials and revise instructions based on their feedback and observations on their performance. If successful, a student should be able to build and add a service to GIFT with a level of effort similar to a class homework assignment.

New agents and services will be integrated either using wrappers developed based on the SuperGLU framework that plug in to GIFT's services (e.g., an adaptor pattern) or by services that directly adhere to GIFT's messaging protocols (e.g., ActiveMQ). For client-side communication (e.g., browser-based services), SuperGLU JS libraries allow rapid integration of multiple cross-domain iframes that communicate using HTML5 messaging and can also communicate with server-side services (e.g., via Websockets).

Service Composition (Selecting and Combining Services)

While programming and adding a single service is something that is straightforward to train for a one-off addition to a system like GIFT, this quickly becomes more complex if you imagine dozens of programmers creating agents which add either new or alternative capabilities (e.g., alternate models to support pedagogical model decisions). This is a service composition problem, where tools must exist to determine which services should be used by a course, to modify default configurations, and to validate that services coordinate meaningfully (e.g., that some other service should be able to send a message required to make another service work effectively). Authoring for agents will rely on the GIFT authoring tools to the greatest degree possible. This requires some level of authoring support in GIFT for Advanced Course Designers, and as such requires careful design consideration before adding such capabilities.

Advanced Agent Capabilities (Ontology Mapping, Hotswapping, Negotiation)

After adding the baseline capabilities for Service design, research will also focus on enhancing the multi-agent communication layer that integrates GIFT, Virtual Human Toolkit, and SuperGLU. The goal is to extend light-weight communication patterns for ITS and other advanced learning technologies. While not all agent communication patterns are appropriate for ITS, a number of existing multi-agent patterns would be particularly beneficial for ITS. The capabilities that we wish to foster in the GIFT ecosystem are:

- *Hot-Swapping:* Adding, removing, or switching services at runtime, which are then discovered by other ITS agents and used (or ignored) appropriately.
- *Service Competition:* Real-time competition between different agents that provide the same information but that might be more appropriate or predictive for different learner types, domains, or contexts (e.g., mobile vs. desktop).
- *Ontology Mapping:* Services that facilitate communication between systems that communicate using messages, but which whose messages use different structures or labels.

Agent communication patterns that can support these goals are agent proposals, agent negotiation, and brokering services, as shown in Figure 2. (Bellifemine et al., 2008). Proposal patterns allow a service to send out a request to a network of agents, who then respond with proposals to do the task (and possibly additional meta-data about performance criteria). The original requesting agent may then select a proposal and submit a firm request to the agent. Proposal patterns can resolve the issue where multiple agents or services can provide the same capabilities, so that a requesting agent needs to prioritize or select one service over another. Negotiation patterns are an extension of proposal patterns, which allow for multi-turn communication to revise a proposal until it meets the goals and requirements of both agents.

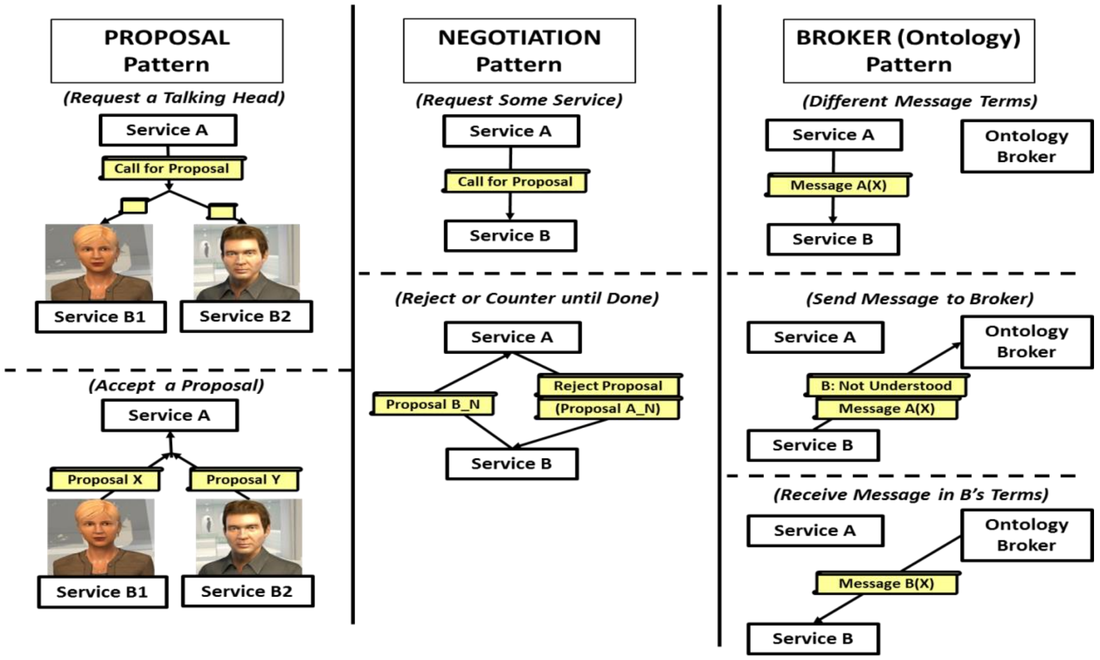

Figure 2: Examples of Key Agent-Communication Patterns for ITS

Brokering agents are a second known-important category of agents, which can be used to store information that is useful to a variety of other agents (FIPA, 2013). The most critical of such agents for Framework Integration is an "ontology broker" to resolve messages would not be understood by some agents or frameworks. An ontology broker stores the semantic messaging definitions and mappings between different ontologies for messages. As such, the ontology broker can attempt to translate a message from a format that the receiving agent cannot understand into one that it can act upon. As part of the first phrase (Framework Interoperability), we have built a prototype ontology converter that focuses on syntactic and semantic mappings between messages from different ontologies (in this case, certain messages shared in common by GIFT,OSVHT, and SuperGLU).

These capabilities would be particularly important for DoD training in future blended learning, augmented reality, and on-the-job training environments. Such environments contain different sensors and different contextual factors (e.g., network availability) that will influence the types of information streams and AI inference that are available. A key benefit of these advanced patterns will be context-based enhancement of training (e.g., taking advantage of capabilities when available) and soft-fail systems (e.g., using lower-fidelity ITS services when more advanced ones are unavailable or ineffective).

CURRENT STATE: FEATURES BEING INTEGRATED INTO GIFT

Work thusfar on the Building a Backbone project has focused on the fundamental problem of Framework Interoperability. In particular, our current progress is intended to simplify two barriers to integrating services and frameworks with GIFT: adding external services and ontology mapping capabilities.

Mapping Between Messaging Ontologies

Mapping between messages from different ontologies is done using a two-stage process. The first step is the identifying that a received message is valid message type to convert into a given target message type (i.e., the expected type for the output of the conversion). In this step, an incoming message checked to determine if it is a valid message type that has been registered in the ontology converter (e.g., a GIFT, SuperGLU, Virtual Human Message, etc.). Once it is ensured that the message is a known type, the system searchers for a Mapping, i.e. a message type that it could successfully mapped onto. This is done using an OntologyMapping object which allows registering mappings, which in this case has registered a limited subset of OSVHT<->SuperGLU and GIFT<->SuperGLU messages that have reasonable equivalents (approximately 15 total for GIFT<->SuperGLU and only one OSVHT<->SuperGLU at present).

In the second step, a series of conversions are applied that transform the original message to the target message type. In some cases, this may be a single conversion (if a direct mapping exists) while in other cases multiple conversions might be required (e.g., in this case, no mappings have been specified for OSVHT<->GIFT, but since certain messages from each ontology have equivalents in SuperGLU, they can be converted through two sequential transforms). The goal of this service is to enable registering semantic message mappings, either at load-time or on-the-fly, that enable information and requests sent from one service or framework to communicate with other services and frameworks. These converters could also be set up as OntologyBroker services which themselves communicate explicitly using messages, to enable agents to resolve messages from remote, centralized converters when needed.

Adding Services from an External Framework

For our first minimal use-case, Virtual Human services were integrated with GIFT by communicating through messages on GIFT's ActiveMQ messaging hub. Under normal conditions, these two frameworks would not communicate because neither sends any messages known to the other framework. To communicate between these systems, a SuperGLU gateway was added to a special Agent Container module in GIFT. This gateway contained an Ontology Converter, as shown in Figure 3. Services in Figure 3 that are built-in to GIFT (i.e., Java services) are shown connected with solid lines, while dashed lines represent connections to services that start up through separate processes. The example case was for the Virtual Human to speak a message from GIFT, using appropriate body-language generated by the OSVHT Non-Verbal Behavioral Generator (NVBG) service and text-to-speech by its TTSRelay service.

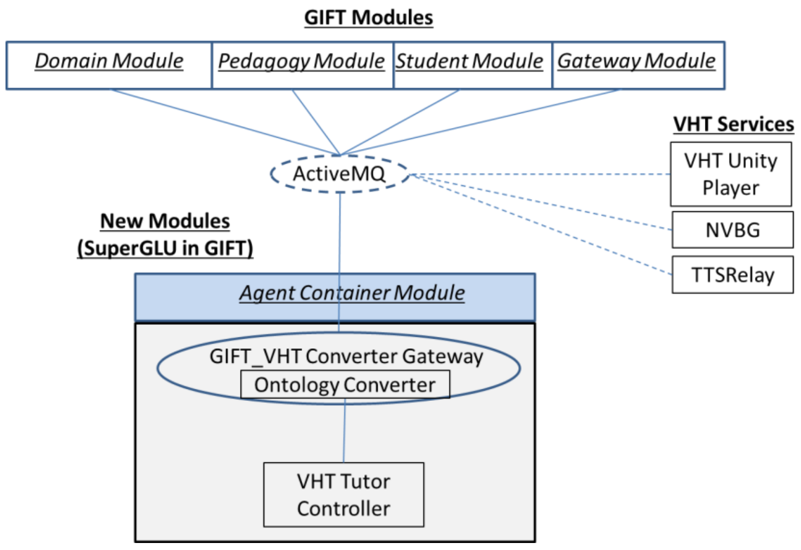

Figure 3. Minimal Integration Example Between GIFT and OSVHT, using SuperGLU

The steps that occur when the message is sent from GIFT to the Virtual Humans toolkit are
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.
While simple, this basic example provides the foundation for more complex service designs.

NEW CAPABILITIES FOR GIFT TUTORING

When considering these additions to GIFT, the question must be asked, "How do these capabilities make GIFT more effective?" While in earlier sections this question was considered at a theoretical and conceptual level, this section considers the advantages and rationale for when GIFT developers should use these capabilities as compared to existing design patterns that GIFT offers. While the focus will be placed on current capabilities being worked on, the next set of capabilities being worked on this year will also be discussed briefly.

The primary rationale the first phrase of this project is to lower barriers to entry for adding new services to GIFT and to demonstrate that functionality by integrating an existing significant framework (the Open Source Virtual Human Toolkit). To understand the benefits of what has been done so far, we must first note the capabilities that already exist in GIFT. Releases of GIFT include three mechanisms for adding external functionality to GIFT: 1) Gateway interop plugins, 2) Domain module conditions that call external assessment engines (e.g., SIMILE; Mall & Goldberg, 2014), and 3) Software branches/forks that modify or extend GIFT as an open source project. Gateway Interop Plugins are the most similar, in that they are designed as interface for external training environments to communicate with GIFT (primarily with the Domain Module). Add-on assessment modules such as SIMILE also have some similarities, in that they represent a submodule plugged in as part of an existing module (in this case, the Domain Module). Finally, at least for GIFT standalone clients (e.g., those not hosted on the GIFT cloud version), new services can be added by simply coding new modules and adding them to a GIFT software build.

In short, GIFT has some established patterns for connecting to other services. Our research here attempts to generalize these patterns. For example, while various training systems can connect as Gateway Interop plugins, these plugins are expected to provide messages that are handled by the Domain Module or by a hard-coded system like SIMILE that provides more fine-grained metrics based on training messages.

There is currently no straightforward way for a training system pass data through GIFT to arbitrary third-party service(s) for further analysis before returning that information to GIFT in an actionable state. In existing GIFT builds, it is unclear where to put novel services that process non-standard messages (e.g., non-GIFT messages, new GIFT-formatted messages) that do not always receive or produce GIFT messages. Second, a system like SIMILE that acts as a subcomponent of a module is currently an explicitly-coded option, rather than part of a larger standardized system of services that allow a module to delegate functionality. Finally, existing GIFT modules assume that they communicate either explicitly through Active MQ messages or sometimes receive information implicitly through other mechanisms (e.g., HTML page choices and submissions). However, as GIFT expands, it will need to pass messages through other protocols to services, such as between other web services and clients (Websockets, REST) or even between HTML iframe services (HTML5 postMessage).

The multi-agent capabilities are being added to GIFT address these challenges, providing new opportunities for developing services for GIFT:

1. AgentContainer Module for Novel Services: First, this work adds a new module to GIFT called the AgentContainer. While this container has no innate functionality, it is special in that it listens to all messages that pass through the ActiveMQ channel so that attached services can process messages and send responses. This container can support services that do not naturally fall into the capabilities of any particular module, including both built-in GIFT services and remote services that are linked in to GIFT as services.

2. Agents in GIFT Core Modules: Second, agents are being made available to add to existing GIFT modules to provide extra capabilities and functionalities. While certain elements of this system are still being designed, the goal would be that these agents would only have access to the same information of the module as a whole, but might override or (in the case of multiple agents) suggest different options to the same situation. This functionality would give a general pattern for implementing certain existing capabilities, such as SIMILE for the Domain Module or work to add reinforcement learning policies to optimize decisions by the Pedagogy Module (Rowe, Frankosky, Mott, Lester, Pokorny, Peng, & Goldberg, 2016).

3. Expanding Gateway Protocols: The SuperGLU library gateways integrated into GIFT are designed to support general-purpose messaging across different protocols. So then, while the existing Gateway Interop Plugin module will still handle connections to many training systems (particularly simulations which are able to contact an ActiveMQ service), the AgentContainer supports building generic gateways to handle other messaging protocols. Currently, a Websocket protocol for real-time bidirectional messaging with web clients (socket.io) is being implemented that will support message passing to and from arbitrary HTML pages opened by GIFT.

4. Communicating with Dynamic or Third-Party Web Pages: On the web-client side, Javascript libraries for Websocket connections to a server (e.g., GIFT) and HTML5 postMessage cross-domain messaging have already been implemented. As such, this work will expand the ability to interoperate with dynamic web pages and cross-domain applications that integrate multiple services on the client side. This means that GIFT courses could be designed to open arbitrary third-party web pages that communicate with a service in the AgentContainer, to enable interoperability with existing frameworks for authoring unique ITS tasks, simulations, or visualizations.

5. Ontology Mapping: As noted above, the ontology mapping capabilities enable gateways added to the AgentContainer to declare how messages in a certain format can be translated into other formats (e.g., GIFT, SuperGLU, VHT, etc.). These are implemented such that messages can be con-

verted before they are sent to remote systems or after they reach third-party destination. In either case, the same set of declarative mappings can be executed to convert messages.

6. OSVHT Integration: Adding open source VHT interoperability to GIFT may also be useful to future authors. While the current build implements the functionality required to control a pedagogical agent (i.e., non-verbal behavior, text-to-speech control, an animated agent), the VHT framework contains other functionality for natural language processing, dialog controllers, and related functionality. While the current work does not make that entirely trivial (new mappings would need to be created), it establishes the fundamental interoperability that should make future integrations simpler.

7. Expanded Analytics: Finally, while the implications of integrating multiple message ontologies are still being explored, one likely opportunity will be to log messages from a variety of systems in greater detail (e.g., web page event logs, third-party service logs that previously could not be sent to GIFT). Since all ActiveMQ messages in GIFT are logged for later analysis, converting messages from external services into GIFT messages would enable deeper after-the-fact processing, even if those messages only represent generic logging messages. With that said, the benefits of this new information would depend on the goals of the researchers and services logged.

While these capabilities are still being tested, refined, and documented with good examples and tutorials, they represent the first wave of functionality provided by the multi-agent architecture approach. The second wave of functionality will focus on issues related to plug-and-play capabilities (e.g., adding services to a GIFT course elegantly and without dealing with any GIFT souce code), hotswapping (e.g., removing and adding agents at runtime based on certain conditions), and patterns to choose between services that provide the same functionality (e.g., proposals, negotiation).

CHALLENGES AND OPEN RESEARCH QUESTIONS

Fundamentally, the challenge of this work is to leverage relatively complex software design patterns (i.e., agent-oriented services) to make the overall experience and ecosystem of GIFT easier and more versatile. In working toward this goal, a substantial challenge is the large differences in skill-sets and use-cases between distinct user groups and use cases. For some users, substantial effort might be warranted provided it only needs to occur once (e.g., integrating a large third-party framework with GIFT and building a full ontology mapping of its messages to GIFT messages). In other cases, any effort beyond normal course activities may be an impossible barrier to cross (e.g., an instructor with limited preparation time trying to add or modify brief course). As such, any multi-agent design must give flexibility for technical users while remaining robust and invisible when interacting with end-users. This is a non-trivial undertaking that raises the question: how can the complexity of software agents be managed such that agents make GIFT *lighter* (i.e., easier to use and maintain) rather than heavier (i.e., harder to understand and configure). As such, a key question is how much of the agent functionality should be exposed to different user types and how such transparency versus opaqueness can be managed.

A second major challenge is that opening the GIFT ecosystem to distributed services introduces challenges for reliability, security, accountability, and debugging. Reliability is impacted because more services and machines introduce greater opportunities for failure, either due to software, hardware, or network errors. Latency can also be a potential issue for services that are distributed across multiple networks and systems: by lowering barriers to add multiple external services, cumulative latency might start to impact responsiveness or performance. Similar problems might be observed with even agents communicating on the same machine with GIFT, if their performance is slow (e.g., a slow agent reducing overall system performance). Security is impacted because information and messages need to cross potentially multiple systems, frameworks, and services within those frameworks. This introduces questions about how trust and credibility should be managed when sending and receiving messages, since even services within the same framework on the same machine might require different levels of authentication and access (e.g., should only be able to exchange certain message types with GIFT). Accountability is likewise more complex in a distributed context: if a service fails for a course, who should be called to fix it? The Course Designer for including it in their course (and might be using it improperly)? The Service Designer who made the service originally (and maybe did not document a limitation that is failing)? GIFT for allowing the service? These are non-trivial problems that require considering analogous situations that support effective ecosystems of services. Finally, debugging is also complex in a distributed, agent-based environment: it is not always clear which agent is failing and the internals of an agent might not always be available to analyze. Worse, the problematic component may not be fixable (e.g., a service from an enterprise framework that requires backwards compatibility). While these challenges may be a ways down the road, and are in some respects good problems to have (e.g., indicate a growing ecosystem) they are also ones where intelligent early decisions can make a positive impact. As such, there are substantial research problems ahead for this work which impact both the technical and social processes of developing and using ITS.

REFERENCES

Bellifemine, F., Caire, G., Poggi, A., Rimassa, & G. (2008). JADE: A software framework for developing multi-agent applications. *Information and Software Technology, 50*(1), 10–21.

Campbell, J., Core, M., Artstein, R. Armstrong, L, Hartholt, A., Wilson, C.A., Georgila, K.,…& Yates, C. (2011), Developing INOTS to Support Interpersonal Skills Practice, In *IEEE Aerospace Conference*.

Foundation for Intelligent Physical Agents (2013). FIPA ACL message structure specification. www.fipa.org/specs/fipa00061/SC00061G.html.

Franklin, S., Graesser, A.: Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In: Intelligent agents III agent theories, architectures, and languages, pp. 21–35. Springer Berlin Heidelberg (1997).

Hartholt, A., Traum, D., Marsella, S. C., Shapiro, A., Stratou, G., Leuski, A., ... & Gratch, J. (2013). All together now. Introducing the Virtual Human Toolkit In *Intelligent Virtual Agents* (pp. 368-381).

Heylen, D.K.J., Kopp, S., Marsella, S.C., Pelachaud, C., Vilhjálmsson, H.H. (2008) The Next Step towards a Function Markup Language. In *Intelligent Virtual Agents.*

Kenny, P., Hartholt, A., Gratch, J., Swartout, W., Traum, D., Marsella, S., & Piepol, D. (2007). Building interactive virtual humans for training environments. In *I/ITSEC,* 7105 (pp. 174-200).

Kopp, S., Krenn, B., Marsella, S., Marshall, A., Pelachaud, C., Pirker, H., Th´orisson, K., Vilhj´almsson, H., (2006) Towards a Common Framework for Multimodal Generation: The Behavior Markup Language *Intelligent Virtual Agents vol. 4133, pp. 205–217*.

Mall, H., & Goldberg, B. (2014). SIMILE: an authoring and reasoning system for GIFT. In *2nd Annual GIFT Users Symposium*. pp. 33-42.

Murray, K., & Silvers, A. (2013). A learning experience. *Journal of Advanced Distributed Learning Technology, 1*(3-4), 1-7.

Nye, B. D., Graesser, A. C., & Hu, X. (2014). AutoTutor in the cloud: A service-oriented paradigm for an interoperable natural-language ITS. *Journal of Advanced Distributed Learning Technology, 2*(6) 35-48.

Nye, B. D. & Morrison, D. M. (2013). Toward a Generalized Framework for Intelligent Teaching and Learning Systems: The Argument for a Lightweight Multiagent Architecture. In *Artificial Intelligence in Education (AIED) 2013 Workshop on GIFT,* (pp. 105-115).

Rowe, J., Frankosky, M., Mott, B., Lester, J., Pokorny, B., Peng, W., & Goldberg, B. (2016). Extending GIFT with a Reinforcement Learning-Based Framework for Generalized Tutorial Planning. In *Generalized Intelligent Framework for Tutoring (GIFT) Users Symposium (GIFTSym4)* (p. 87-97).

Scherer, S., Marsella, S.C., Stratou, G., Xu, Y., Morbini, F., Egan, A., Rizzo, A., Morency, L. (2012), Perception Markup Language: Towards a Standardized Representation of Perceived Nonverbal Behaviors. In *The 12th International Conference on Intelligent Virtual Agents*

Sottilare, R. A., Goldberg, B. S., Brawner, K. W., & Holden, H. K. (2012). A Modular Framework to Support the Authoring and Assessment of Adaptive Computer-Based Tutoring Systems (CBTS). In *I/ITSEC, vol. 2012* (1). NTSA.

Ventura, M., Hu, X., Nye, B. D., & Zhao, W. (2015). NewtonianTalk: Integration of Physics Playground and AutoTutor using GIFT. In *AIED 2015 Workshop on Developing a Generalized Intelligent Framework for Tutoring (GIFT)* (p. 23-29).

Vilhjálmsson, H. and Thórisson, K.R, (2008) A Brief History of Function Representation from Gandalf to SAIBA . In *Proceedings of the 1st Function Markup Language Workshop at AAMAS.*

ABOUT THE AUTHORS

Benjamin D. Nye, Ph.D. *is the Director of Learning Sciences at USC ICT. He received his Ph.D. in Systems Engineering from the University of Pennsylvania in 2011 and previously served as a Research Professor at the University of Memphis. Ben's major research interest is to identify best-practices in advanced learning technology, particularly for frontiers such as distributed learning technologies (e.g., cloud-based, device-agnostic) and socially-situated learning (e.g., face-to-face mobile use). Ben's research tries to remove barriers development and adoption of intelligent tutoring systems so that they can reach larger numbers of learners, which has traditionally been a major roadblock for these highly-effective interventions.*

Daniel Auerbach *is a Research Programmer at USC ICT, who has developed and expanded a variety of intelligent tutoring systems and frameworks. He received a B.A. in Linguistics from Cornell University.*

Tirth Rajen Mehta *is a Student Research Worker at USC ICT, who is working on his M.S. in Computer Science at the USC Viterbi Department of Computer Science.*

Arno Hartholt *is the Director of Research and Development Integration at the University of Southern California Institute for Creative Technologies where he leads the virtual human integration and central asset production & pipeline group. He is responsible for much of the technology, art, and processes related to virtual humans and related systems, in particular at the interchange between research and industry capabilities. He has a leading role on a wide variety of research prototypes and applications, ranging from medical education to military training and treatment. Many of the ICT virtual human capabilities are freely available to the academic research community and US Military through the Virtual Human Toolkit (vhtoolkit.ict.usc.edu). Hartholt studied computer science at the University of Twente in the Netherlands where he got his Master's degree.*