MissionEngine: Multi-system integration using Python in the Tactical Language Project

Hannes Vilhjalmsson, Prasan Samtani

Center for Advanced Research in Technology for Education
University of Southern California – Information Sciences Institute
{hannes, samtani}@isi.edu

ABSTRACT

The Tactical Language Training System (TLTS) aims to help learners to rapidly acquire basic competence in a foreign language and culture. Trainees work through lessons in a schoolhouse environment; after which they must carry out specific missions in a simulated world by interacting with virtual characters through a speech recognition interface. TLTS contains several medium and large sized components, some of which derive from earlier research projects. This paper presents the challenges faced when trying to integrate the various component technologies into a single architecture, and describes the positive role that the Python language has played in the process.

INTRODUCTION

The Tactical Language Training System is an educational game designed to help learners develop competence in a foreign language and culture, organized around specific communication tasks. Learners work through vocabulary and cultural lessons in a schoolhouse environment. An intelligent tutoring agent is employed to monitor learner progress, and feedback is provided based on learner pronunciation (assessed using a speech recognition interface) and history. Learners must then practice their skills in a game environment, where they must interact with AI characters using speech and gesture in order to complete story driven objectives and advance levels.

The system makes use of several component technologies, some of which are derived from earlier research projects. We begin by describing the interactions within TLTS, and the role that the different components play. We then describe the overall architecture of TLTS, and explain the motivations behind our architectural decisions. We conclude by presenting a strong case for Python as a language for integration, and examine its impact on our project's development, authoring, distribution, and maintenance.

There are three primary interactions within the Tactical Language system. The Mission Skill Builder (MSB) is a set of interactive exercises where learners are introduced to the vocabulary and pronunciation of the language [1]. A screenshot of the MSB is provided in Figure 1. Learners can practice listening to and recording various phrases as they appear in lessons. The recordings are then analyzed by a speech

recognizer, and passed the processed recording to an intelligent tutoring agent, which provides socially appropriate feedback based on disfluency patterns and learner history [2,9].

Figure 1: A screenshot of the MSB

The Mission Practice Environment (MPE) is a story-based game environment in which learners must use their newly acquired linguistic and cultural skills in order to accomplish missions and advance levels. Learners must speak to virtual characters (using the speech recognition interface) in order to accomplish mission objectives, and they must use the appropriate phrases and gestures in order not to lose the trust of the characters. For example, in the current system, learners are given the task of early postwar reconstruction in the fictional Iraqi town of Al-Wardiya. In order to succeed in missions, they must learn how to greet people, introduce themselves, and ask for and follow directions. A screenshot of the trainee greeting a civilian in the MPE is found in Figure 2.

The Mini-game is a third mode of interaction, and is a simpler speech-based arcade game in which learners can practice listening to and speaking shorter phrases. TLTS is implemented as a Total Conversion Mod to Unreal Tournament 2003, although the backend is almost exclusively written in Python.

Figure 2. Screenshot of the MPE

ARCHITECTURAL COMPONENTS

TLTS employs several components, some of which are external systems, and others which are ongoing research projects. This section introduces the various components and provides a brief description of each.

Gamebots [3] is an interface that allows Unreal Tournament bots to be controlled via network sockets connected to clients (available in several languages, including C++, Java and Python). Gamebots is the result of a previous project that was a collaboration between USC/ISI and Carnegie Mellon University and is publicly available at http://www.planetunreal.com/Gamebots

The Pedagogical Agent is a socially intelligent agent that provides feedback and encouragement to the learner based on pronunciation quality, learner history, and the inferred motivation of the user. The Pedagogical agent is implemented in Python [2].

The ASR (Automatic Speech Recognizer) is a speech recognition system built on top of the Hidden Markov Model Toolkit (HTK), initially developed at Cambridge University, and modified for the Tactical Language Project by the Speech and Image Processing Lab at USC. The ASR is implemented as a C++ library [9].

PsychSim is a multi-agent system used for modeling social cognition in individuals and groups [4]. It is the underlying decision-making framework of the virtual characters in the MPE. Characters possess their own goals and motivations, and possess a set of beliefs of the world around them. PsychSim is implemented in Python.

SocialPuppets is a module that realizes physical character behavior in the environment given a description of the character's intent from PsychSim. Together, PsychSim and SocialPuppets for the 'character AI' form the entities in the MPE. SocialPuppets is implemented in Python, but has a corresponding module responsible for lower level behaviors (UnrealPuppets), which is implemented UnrealScript – a proprietary scripting language employed by the Unreal Game Engine.

The DataManager is a persistent storage module that is used for all data storage in the system. It is implemented in C++ as an XML database using Berkeley DB.

ARCHITECTURE

Integrating and coordinating the modules presents a significant challenge, as there are several factors to consider when designing an architecture. Speed, flexibility, robustness, reliability, and the ability for the system to run with one or more modules absent are all important factors. Earlier versions of the system employed a classical messaging architecture using a Elvin, a freely available messaging service, but problems with latency and unreliability forced us to consider other options. Our current architecture is presented in Figure 3., the MissionManager and SkillBuilderManager modules are coordinators of the dataflow between modules.

Inter-module messaging was slowly removed from the earlier system, primarily due to synchronization issues and concerns about latency. Flexibility had to take a backseat to speed and robustness in this case, although carefully defined module boundaries, and the use of a dynamically typed language helped alleviate any concerns that the system would be too rigid and too slow to respond to design changes. For C and

C++ components, we wrote Python extension modules (SWIG for the DataManager, hand-coded extensions for the ASR), which significantly reduced latency caused by messaging. The Pedagogical Agent, previously implemented in Java, is being rewritten using Python.

Messaging still exists in the current system in the form of GameBots, which is required to communicate with UnrealScript modules; but this has been limited to AI-level behaviors, script triggers and user-interface events.

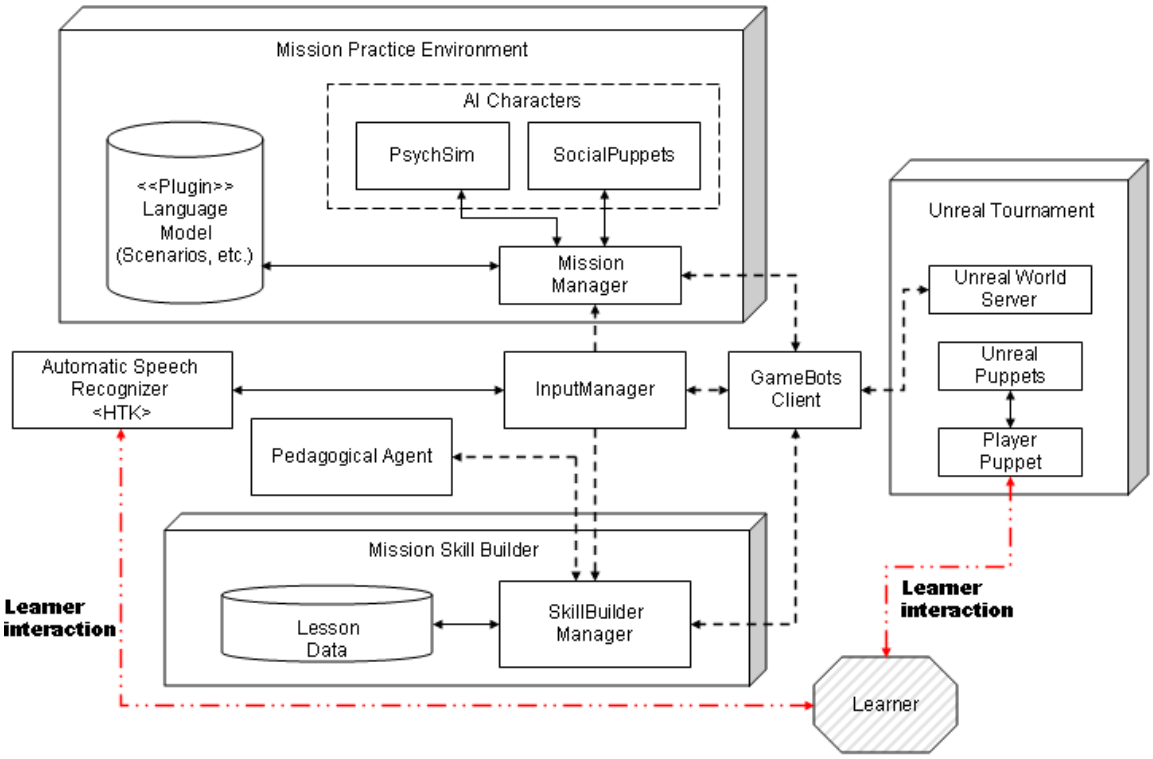

Figure 3. MissionEngine Architecture

THE CASE FOR PYTHON

As our system is a research prototype, change is inevitable – even more so than in the commercial domain. Even compared to the average research project, TLTS can be considered to consistently be in a state of great flux. This is partly due to the fact that Tactical Language has been developed highly iteratively, and been evaluated using an iterative formative evaluation process [7] – there have been six different evaluations thus far. Each evaluation brings suggestions, proposals, complaints, and great change. Flexibility is a requirement, but we could not rely on the traditional message-passing architecture used for flexible AI systems, because of issues with latency and coordination – both of which can destroy the interactive experience. Luckily, flexibility was available through the choice of our programming language.

It is often the case that the programming language is chosen as an afterthought, or maybe, as in our case, no single programming language is chosen at all. Earlier versions

of TLTS employed a messaging service called Elvin for inter-module communication, and the availability of clients in several languages meant that developers could program in whichever language they were most comfortable in. This led to an explosion in the number of languages employed by the system – Java, C++, Python, UnrealScript were all used simultaneously. Frustrations with system reliability, latency, and bloated installation requirements led to the removal of Elvin and Java from the system, and Python extension modules were written for the remaining C++ modules. While we didn't plan this at the start, the majority of our code (save for speed-critical animation and speech recognition code) is written in Python.

Python fit our project perfectly – instead of attempting to reduce the entropy in our project, we simply moved toward a language that understood it. Dynamic typing meant we didn't have to change class definitions every time the data formats changed, or when new features were requested. The availability of standard modules allowed the rapid production of authoring tools. The pickle module allowed for a pleasantly easy form of binary XML – files are parsed at compile time and the DOM tree is serialized using pickle. This reduced our latency when loading lessons from two minutes to around five seconds!!

The importance of standard and external modules in our system cannot be understated. *Tkinter* is used for configuration scripts, the *os* and *shutil* modules are used for building the package, *socket* was used for implementing the Gamebots client, and the use of *PyXML* and *pickle* is ubiquitous. The software is built using *py2exe*, as several of our customers (most often military bases) forbid the installation of external software.

We have found Python to be an excellent choice as a language for integration, the inherent flexibility and the oft repeated virtue of *batteries included* certainly proved to be the difference between success and failure in our case. Figure 4 summarizes how Python contributed to the successful fulfillment of our TLTS requirements.

Requirement	**Python Solution**
Flexibility	The dynamic nature of python allows for latent typing [10], in our case, this is more than just syntactic sugar. In UnrealScript, we had to create a new class for every single page type in the skillbuilder, in Python, there is a single page type, attributes are dynamically instantiated.
Distribution / Building	Distribution is a bigger problem than it appears. Several of our clients (most of them military) refuse to install certain software packages. The ability to bundle Python with py2exe makes distribution much easier, and simplifies the installation process.
Cross-compatibility	Compatibility with C++ is another major plus. Both the interfaces to the ASR and the upcoming DataManager rely on SWIG and the extension interface.
Data-drivenness	The availability of standard module packages such as PyXML makes parsing data easy, and serialization modules like pickle and shelve work together to make our version of "binary XML" possible.

Figure 4: Pythonic solutions

SUMMARY

Large scale applications face a different set of problems than smaller ones. Changing requirements, evolving component interfaces, and constant design and content changes make flexibility a must. Using the Tactical Language Training System as an example, we show that flexibility in a programming language can sometimes overcome the inflexibility of a typical monolithic architecture; and that the choice of programming language is important, one that cannot be decided simply based on the comfort level of project developers – after all, none of us knew any Python before we started.

ACKNOWLEDGEMENTS

This project is part of the DARWARS Training Superiority Program of the Defence Advanced Research Projects Agency. The authors of this paper would like to acknowledge the contribution of W.L. Johnson and the Tactical Language Team. We would also like to thank DARPA, the University of Southern California, and everyone who participated in our evaluations.

REFERENCES

- [1] Johnson, W.L., Vilhjálmsson, H., and Marsella,S. 2005 Serious Games for Language Learning, Artificial Intelligence in Education
- [2] Johnson, W.L., Wu, S., & Nouhi, Y. 2004 Socially intelligent pronunciation feedback for second language learning. ITS ’04 Workshop on Social and Emotional Intelligence in Learning Environments.
- [3] Adobatti, R., Marshal, A., Scholer, A., Tejada, S., Kaminka, G., Schaffer, S., and Sollito, C. 2001 Gamebots: A 3D Virtual Test Bed for Multi-Agent Research
- [4] Marsella, S., Pynadath, D., Read, S. 2004 PsychSim: Agent Based Modelling of Social Interactions and Influence
- [5] Hayes-Roth, B., 1985 A blackboard architecture for control
- [6] Swartout, W., Gratch, J., Hill, R., Hovy, E., Lindheim, R., Marsella, S., Rickel, J., Traum, D., 2003 Simulation Meets Hollywood: Integrating Graphics, Sound, Story and Character for Immersive Simulation
- [7] Johnson, W. L., Beal, C. 2005 Iterative Evaluation of an Intelligent Game for Language Learning
- [8] Johnson, W.L., Vilhjálmsson, H., & Marsella, S. (2004). The DARWARS Tactical Language Training System. Proceedings of I/ITSEC 2004
- [9] Mote, N., Sethy, A., Silva, J., Narayanan, S., Lewis Johnson, W.L. (2004). Detection and modeling of learner speech errors: The case of Arabic tactical language training for American English speakers. InSTIL 2004, in press
- [10]Eckel, B. 2004 About Latent Typing - Thinking about computing (http://www.mindview.net/WebLog/log-0051)