2020-04-07

From Systems to Services: Changing the Way We Conceptualize ITSs -- A Theoretical Framework and Proof-of-concept

Brice R. Colby
Brigham Young University

From Systems to Services: Changing the Way We Conceptualize Intelligent Tutoring Systems

A Theoretical Framework and Proof-of-Concept

Brice R. Colby

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Instructional Psychology and Technology

Brigham Young University

ABSTRACT

From Systems to Services: Changing the Way We Conceptualize Intelligent Tutoring Systems
A Theoretical Framework and Proof-of-Concept

Brice R. Colby
Department of Instructional Psychology and Technology, BYU
Doctor of Philosophy

This dissertation consists of two articles. The first article describes an architecture for intelligent tutoring that focuses on modularity. This new architecture is based on Gibbons' layers theory for instructional design (2014). Splitting up the architecture for an intelligent tutor into layers allows different pieces to age at different rates which, in turn, allows the intelligent tutor to be adapted to new research and design theories. This architecture supports building intelligent tutoring *services*, nimble programs that can be assembled together to replicate the functions of intelligent tutoring without the expertise needed to create the services. Alternative architectures support building intelligent tutoring *systems*, monolithic programs that are less amenable to change and require immense expertise.

The second article provides a proof of concept for the first services created under the layers theory. These two services create the building blocks of a domain and comprise one part of the content layer as described in the first article. The first service focuses on the task of key concept extraction whereas the second service focuses on prerequisite relationship extraction. These two tasks can provide the structure of the domain, particularly when it comes to domains that are more declarative in nature rather than procedural.

Keywords: intelligent tutoring, intelligent tutoring services, layers theory

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DESCRIPTION OF RESEARCH AGENDA AND STRUCTURE OF DISSERTATION

This article-format dissertation explores the development of intelligent tutoring systems (ITSs) from the perspective of a modularized framework. Nye (2015) has argued that the age of monolithic, stand-alone systems has passed, and a new age of intelligent tutoring *services* needs to begin. While Nye makes a reasonable argument as to why authors of ITSs should adopt a services-based design approach, he stopped short of defining a framework to guide design decisions and to facilitate the creation of coordinated services. Borrowing Gibbons' layers theory for instructional design (2014), I propose the layers theory as a framework for guiding the development of intelligent tutoring *services* rather than intelligent tutoring systems. Service-based approaches modularize the components of a system such that they are interchangeable and can be independently developed and maintained.

This dissertation consists of two articles that build upon previous work from my master's thesis and a PhD project. To situate the need for these two articles, I first discuss the previous work I have done in consolidating 25 years of ITS research and development. I then discuss my attempt at applying what I learned to make an online learning environment more intelligent by creating a feedback engine. The difficulties I encountered while designing this feedback engine prompted me to recognize the validity of Nye's argument and the need for a framework to assist in the design and development of ITSs. I then present two articles.

The first article is a theoretical framework that demonstrates how Gibbons' layers theory can be applied to create intelligent tutoring services. I draw on extant ITS literature to demonstrate how each layer can successfully inform the design of intelligent services.

For the second article, I apply the intelligent services framework as a guide to develop a service for the content layer. This service uses natural language processing and open textbooks to

create ontologies for different subject matters, a vital first step in the development of an ITS. Article two describes the design, development, testing, and refinement of these intelligent tutoring services.

Together, these two articles add an important, but missing, component to the ITS community of researchers. Specifically, the theoretical framework (Article 1) provides a guide for the development of a range of intelligent services. Then, to demonstrate the feasibility of this framework, I developed and researched a content-based service for ITS, as informed by Gibbons' layers theory (2014).

PREVIOUS WORK

My master's thesis was an extensive review of ITSs from 1990-2015 (Colby, 2017). Nearly half of the 800+ articles I found were dedicated to 10 ITSs, which then became the focus of my review. Using a standard framework as a lens to understand the design of ITSs, I summarized problems and solutions related to ITS development, and identified trends and patterns in ITS designs.

Major trends and patterns aligned with the four-component framework that served as the lens for the paper: domain model, tutor model, student model, interface. My analysis revealed a major gap in the way ITS development literature, which is added as a 5th component—learning gains. The first component, the *domain model*, was largely focused on science, technology, and mathematics as subject domains for ITSs. The second component, the *tutor model*, relied on constructivism as the theoretical strategy to inform tutoring decisions. The tutoring tactics used in ITSs stemmed from this strategy. The third component, the *student model*, describes the many ways that an intelligent tutoring system infers what a student knows. In the paper, I described the wide variety of data and techniques that have been used to model student knowledge. The fourth component, the *interface*, revealed that most ITSs have shifted to the web, but vary in their capacity to interact with students. I also found that user experience is not heavily reported on and ought to be included more in future research. Lastly, I added an important fifth component, *learning gains*, as a measure of comparing the effectiveness of ITSs and their designs. Overall, the research shows that ITSs are capable of producing learning gains equivalent to a human tutor. However, like user experience, learning gains are not heavily researched or reported in the literature. The full thesis can be accessed at: http://hdl.lib.byu.edu/1877/etd9693

This framework and my research provided a side-by-side comparison of design decisions that prompted me to take a common component of ITSs—feedback—and attempt to add feedback to MyEducator, an existing online learning environment used extensively by the Business School at BYU. MyEducator provides a course for learning Microsoft Excel. As part of the course, students solve problems in practice assignments. These practice assignments did not provide feedback to the student until after the assignment was submitted and graded. A hallmark of ITSs is just-in-time feedback as a tutoring option (Colby, 2017). I created a feedback engine that provided two forms of just-in-time feedback to students as they were solving problems. The first form of feedback is flag feedback, or a simple demarcation if the solution is correct or incorrect. Additional feedback was provided to the student through a gradated hint system delivered to students by request.

Development of the feedback engine was labor intensive. Even though the rules for the engine were the same for each problem, they had to be tailored to fit each problem and provide relevant hints. While the feedback engine seemed to have positive short-term effects on student performance, scalability and portability issues prevent the engine from being more useful.

My own experience is congruent with the literature on ITS development. Estimates put development time for one hour of instruction between 100 and 1000 hours without authoring tools. Some authoring tools have reduced that development time substantially to estimates of 30-40 hours for one hour of instruction (Mendicino, Razzaq, & Heffernan, 2009). My intuition is that intelligent tutoring *services* can reduce the overall development time and cost of an ITS. The remainder of this dissertation lays out a framework for the design of Intelligent Tutoring services and then details the development and validation of an intelligent tutoring service for the content model as a practical proof-of-concept to the theoretical framework

ARTICLE I

From Systems to Services: Changing the Way We Conceptualize Intelligent Tutoring Systems

Brice R. Colby

Peter Rich

Department of Instructional Psychology and Technology

Brigham Young University

150 MCKB – BYU

Provo, UT 84602

Abstract

This paper describes a framework that focuses on a modular architecture for intelligent tutoring systems (ITSs). The benefit of this framework is to conceptualize ITSs as an ecosystem of services rather than standalone systems. Seven layers make up the framework with each layer representing an aspect of intelligent tutoring. Within the seven layers are several sublayers that represent services that could be created under this framework. The seven layers and their sublayers are: content (knowledge representation and learning objects), data management (data storage, student model, tutor model, detectors), strategy (micro-adaptation, macro-adaptation, meta-adaptation), control (domain, generic, instructional, and system controls), message (strategy, content, and data communication), representation, and media-logic (communication between layers, delivery method, and authoring tools). This framework outlines the structure needed for creating an ecosystem of intelligent tutoring services.

Keywords: intelligent tutoring systems, modularity, intelligent tutoring services, intelligent tutoring framework, artificial intelligence in education

From Systems to Services: Changing the Way We Conceptualize ITSs

The promise of having a personal tutor for each student has driven intelligent tutoring research for decades. Researchers and developers of intelligent tutoring systems (ITSs) envisioned ITSs as “powerful, flexible systems that adapt in a range of ways to the learner” (Baker, 2016, p. 608). Indeed, much progress has been made to get systems to a place where they are comparable to human tutors (VanLehn, 2011). However, Baker (2016) argues that the ITS leaders (i.e., in terms of systems that are widely used) represent a field that relies less on artificial intelligence and more on leveraging human intelligence via intelligently designed systems. That is, *Stupid Tutoring Systems, Intelligent Humans* (Baker, 2016). Furthermore, Baker clarifies that the goal of artificial intelligence in education (AIED) is “not to promote artificial intelligence, but to promote education” (Baker, 2016, p. 610).

While Baker’s (2016) article points to the dissonance between ITSs at scale and the original vision of intelligent tutoring in an effort to suggest a new vision for the field, that dissonance is symptomatic of inflexible architectures rather than a mismatch of vision. The architectures or designs of ITSs at large inherently limit an ITS’s ability to adapt. Singular, proprietary systems do not contribute to a cumulative knowledge in the field. Instead, they become silos of research that may have tangential impact on others’ designs and research.

This is not a new critique. Vassileva (1990) pointed to ITSs’ strong domain dependence and inflexible structure as a reason ITSs are not widely adopted in schools despite repeated evidence of their effectiveness. Nkambou, Bourdeau, and Psyche (2010) echoed a similar concern regarding paradigmatic research silos that don’t contribute to sharing. They went so far as to say, “As a result, after thirty years, existing solutions are still not widely shared in the field, making it difficult to find adequate building blocks and guidance to build an ITS” (p. 11). Instead of adopting a new vision, we first need to look at the structures in place that limit our ability of designing “powerful, flexible systems that adapt in a range of ways to the learner” (Baker, 2016, p. 608). We need to change the way we conceptualize ITSs from independent systems to an ecosystem of services.

The argument for an ecosystem of intelligent services is not a new one (Brooks, Winter, Greer, & McCalla, 2004; Brusilovsky, Sosnovsky, & Shcherbinina, 2005). More recently, Nye (2015) made the argument again that ITSs are heading to a services-based ecosystem and points to other web services as examples of what services look like. For example, a basic blog site can use authentication systems from Facebook or Google, ads can be targeted to specific audiences, and media can be embedded from anywhere on the internet. These services are owned and maintained by one but used by many, reinforcing the idea that no single application needs to do everything anymore. Furthermore, the addition of services enhances the overall capabilities of a specific software solution. The same can and should be said of ITSs, especially because they are moving to web-based delivery platforms. This indicates a shift from the traditional architecture of an ITS as an all-in-one, closed system to that of a service-oriented architecture.

Education has already seen the benefit of service-oriented architectures. Thanks to standards like Learning Tools Interoperability (LTI), content can be created and shared on any LTI-capable platform regardless of the institution’s learning management system (LMS). An educator using the Canvas LMS (n.d.) has access to a multitude of services, which can extend the LMS well beyond its native capabilities. This allows courses to be customized per their needs without adding unnecessary elements. For example, educators can use services like Delphinium as an add-on to the educational experience. Delphinium is an LTI-compliant framework for hosting gamification components that address learner motivation (Delphinium, n.d.). It allows an educator to add elements like a leader board, achievements, or easter eggs to their online learning environment. Without the LTI standard, a service like Delphinium would need to be custom built for each course or LMS. If ITSs were to adopt a similar approach, the idea of a traditional four-component architecture (Wenger, 1987) would become obsolete in the sense that an ITS would need to be designed from the ground up with all four components as a monolithic, stand-alone system. Rather, the components could be provided by services that are neither built nor controlled by the authors. This services-based approach would enable the development of more and more customized ITSs, streamlining and simplifying the ITS development process.

However, the traditional ITS architecture is not nuanced enough to provide direction on the types of services that should be developed. For example, the tutoring model, or the pedagogy, of an ITS *can* include messages to the student in the form of feedback. It *can* include the strategy and tactics used to navigate problem selection or timing of interventions. It *can* dictate whether or not the student should use free responses or answer multiple choice questions. Traditional ITS components *can* include a lot, but are not explicit enough to guide the creation of services, and, as such, they have become a panchreston in a new age of ITS development.

For years, ITS development has used the 4-component model (Wenger, 1987) as the primary developmental framework. While this model is still relevant today, ITS development needs a framework that positions modularity at the center of the design process. Attempts have been made to provide a modular framework, the GIFT architecture being one such example (Brawner, Goodwin, & Sottilare, 2016). One of the drawbacks of GIFT is that it still relies heavily on the traditional architecture of ITSs, making it difficult to inform the creation of new services. While the question of how granular a framework should be is arguable, Gibbons' layers theory (2014) provides enough granularity to be useful, and it was created with the idea of modularity from the outset, thus typifying a service-oriented architecture.

Design Layers

Gibbons' (2014) layers theory is based on an architectural approach to instructional design (architecture here referring to buildings and homes rather than computer software). The theory emphasizes the value of looking at instructional products as parts that age at different rates much like a house does. The advantage of this modular approach is that when one part of the system becomes deprecated, it can be updated or replaced with minimal impact on the system as a whole. ITS developers (or even end users) can upgrade or add parts without interrupting the overall service. Layers are analogous to intelligent tutoring services. The theory consists of seven layers: message, control, representation, content, strategy, data management, and media-logic (see Figure 1). Gibbons emphasized that “these layers are derived on functional grounds; they represent functions carried out by virtually every instructional artifact” (2014, p. 34). A description of each layer follows.

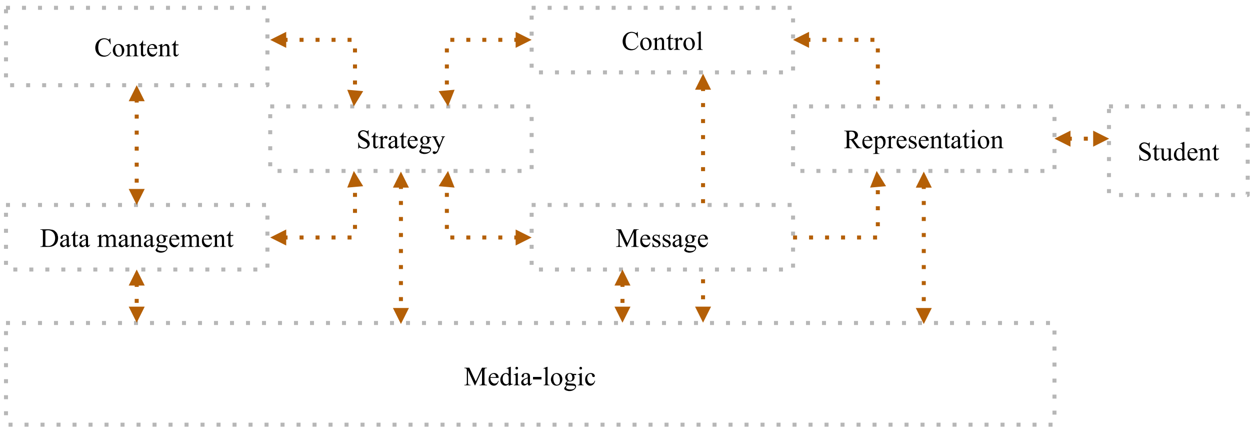

Figure 1. The layers and their relation to each other. Adapted from Gibbons (2014).

The *representation* layer is the only tangible layer of the design. It comprises the visual interface of the design as its role is to convey the sensory experience to a student. Often, the representation is so entwined with the rest of the instructional product that a change in representation requires a change in the entire product. Considering representation strategies are constantly changing, evolving, or fading into obsolescence, it becomes imperative to create this layer as independently from the others as possible.

The *control* layer describes the ways a learner has to communicate with the instruction as well as take action. This can be considered part of the interface of the tutoring system and can include student input and navigating the instruction. In other words, it allows the student to express their desires to the system. The benefit of conceptualizing the control layer as a separate part of the interface is that it becomes easier to change the modes of communication the student has with the system.

The *message* layer describes the way instruction communicates with the learner. Together, the control and message layers create a two-lane highway where instruction and student communicate back and forth. The message layer should be considered distinct from the strategy layer, which dictates higher-level goals, and the representation layer, which gives form to the message layer. The message layer's aim is to turn the high-level goals of the strategy layer into conversational patterns. The use of flag feedback, or the marking of the answer as correct or incorrect, is an example of the message layer. This is not to be confused with the representation of the message. The representation of the feedback can come in many forms like colored text or symbols to show correctness. This demonstrates that while the message layer is singular in purpose (i.e., fulfilling the strategy's high-level goal), it can be represented in many ways, sometimes simultaneously.

The *strategy* layer acknowledges that instructors and learners both have goals. Matching goals between instructor and learner becomes a priority of this layer. Gibbons (2014) describes three levels of strategic goals that require alignment: instructional, strategic, and means. The instructional goal describes the outcome of the learning experience. The strategic goal describes how a learner will meet the instructional goal. Finally, the means goal describes the specific actions needed to reach the strategic goal. This goal structure highlights the relationship between the strategic layer and the other layers. Gibbons argued that "all of the other layers represent an extension of the strategy layer" (2014, p. 41). While there are many instructional strategies to choose from, Gibbons said that "strategy itself can only be properly understood as a dynamic process of shifting responsibility over time from the instructor and designer to the learner" (Gibbons, 2014, p. 41). Understanding instructional strategy from this perspective, we can start to see that no one instructional theory can meet all needs at all times. Instead, instructional strategies can and should be adapted to meet instructional goals from the perspective of both the designer and the learner.

The *content* layer refers to abstract knowledge structures and is not to be confused with the representation layer or with media resources. Rather, this layer is "concerned with the nature and structure of the knowledge to be learned and its capture in a form that can be used during design and delivery of instruction" (Gibbons, 2014, p. 43). In the traditional domain model, the content layer has been captured as ontologies, constraints, or production rules (Colby, 2017). However, this layer is not concerned solely with knowledge regarding the subject-matter. The instructional theory of cognitive apprenticeships lists additional knowledge such as problem-solving strategies, problem-solving heuristics, and learning to learn (Collins et al., 1989) which can be part of the content layer. Some authors have developed mini-tutors that focus on these metacognitive skills ranging from help-seeking behavior (Roll, Aleven, McLaren, & Koedinger, 2011) to emotional awareness (Arroyo, Mehranian, & Woolf, 2010).

The *data-management* layer defines how data is “gathered, remembered, analyzed, and how it is used beneficially” (Gibbons, 2014, p. 44). This layer most aligns with the traditional student model of an ITS; however, it is broader in scope to include all forms of data management that is required. Data analyses can be distributed to other layers for decision-making processes (e.g., sending data to the strategy layer to determine the next sequence of instructional events).

The *media-logic* layer is what executes all the other layers at the same time and can be considered the delivery method of the layers. The traditional media-logic of instruction is a human teacher. A teacher can bring together different instructional strategies, keep data on a student, create a mental structure of the content, etc. A technological solution is a little more complex. There are two parts to consider. First, the delivery platform. This could be the web, a mobile app, or even a robot. The second is the communication between layers. This can be standards or a framework that facilitate the conversation.

Layers as a Service

In order to reify how the layers theory can be applied to ITS development, this section breaks down each layer in greater detail and provides examples of services from the ITS literature that fulfil the function of that layer. In the absence of any examples in the literature, I describe a theoretical service and how that would operate in the larger ecosystem of layers and services.

Content Layer

The content layer, which addresses the structure of knowledge, aligns most closely with the standard domain model of ITSs. To cover what is typically included as part of the domain model, the content layer is composed of two sublayers: knowledge representation and learning objects.

Knowledge representation. Regardless of the methodology, ITSs need to capture the knowledge to be taught. The knowledge representation is what enables the ITS to make tutoring decisions like sequencing or to model student mastery of concepts in the domain. Knowledge has been captured in many ways by ITSs with common methods including ontologies, production rules, and constraints (Colby, 2017). While the subject matter is generally captured by these methods, additional knowledge can be represented by these methods like buggy knowledge (i.e., misconceptions) or metacognitive skills (Arroyo et al., 2010; Roll et al., 2011). When deciding how to capture the domain knowledge, one must consider the nature of the knowledge. For example, ontologies deal mostly with declarative knowledge. Constraints or production rules might be better suited to represent procedural knowledge. In some cases, it may be necessary to combine the two in order to provide a more robust content layer. The ontology can represent declarative knowledge while also providing the relationships needed for sequencing decisions. Layered on top of that could be production rules or constraints that can handle the problem-solving knowledge of the domain. Even in domains like history that don't have explicit rules, there is still problem-solving knowledge existing in "virtually every subject-matter area: (1) 'rules of thumb' or heuristic strategies used in solving problems, and (2) rules for selecting a heuristic to be used in a particular problem-solving situation" (Gibbons, 2014, p. 290). These rules are better represented as production rules or constraints, which further emphasizes the symbiotic relationships between ontologies and rules.

Knowledge representation is a bottleneck in the development of ITSs (Moundridou & Virvou, 2003). There have been attempts to apply machine learning techniques to make the process (semi-)automated. An example of learning production rules, JESS was designed to automatically induce production rules with machine learning (Jarvis, 2004). JESS was successful in generating rules for multiplication, fraction addition, and tic-tac-toe by using examples. JESS learned the rules in less than a minute, but the learning required brute force, which resulted in short rules that were too general at times.

The Constraint Acquisition System (CAS) also learned from examples to semi-automatically create constraints (Suraweera, Mitrovic, & Martin, 2005). Here, a domain expert creates an ontology of the domain. CAS can automatically generate syntax constraints (i.e., restrictions on structure) from the ontology. The expert can then provide problems and solutions to CAS to learn semantic constraints (in this case, being able to compare an ideal answer to alternate correct solutions). Finally, the expert validates the constraints. CAS was tested against the constraints manually developed for KERMIT, an ITS for database modeling. CAS generated constraints that covered 85% of those found in KERMIT. After analyzing the CAS's constraints, the authors said that CAS did not cover more of the constraints due to the lack of examples; only six examples were used for CAS. Considering that such few examples were used, CAS seems to demonstrate an effective way to generate constraints and could be more effective with additional examples.

As for generating ontologies, many services already exist that use natural language processing to semi-automatically generate ontologies (Ghorbel et al., 2016). A popular, open-source project, Text2Onto was used recently in a computer science class to generate the important concepts from a textbook (Ismail, Ahmadon, & Shahbudin, 2018). In this paper, Text2Onto was able to automatically extract 139 concepts from 92 pages (the first chapter) of the course textbook. However, the representation of the ontology needed to be done manually by domain experts since the relationships between concepts is not generated during the concept extraction. Protege was used as the ontology editor in this case (Protege, n.d.).

Learning objects. The basic unit of the content layer is a learning object. Merrill (1983) notes in his component display theory that a basic learning unit, or a learning object, includes facts, concepts, procedures and principles. Reigeluth, Merrill, and Bunderson (1978) described four relationships that can exist between learning units: requisite relationships, conceptual relationships, procedural relationships, and theoretical or principles-based relationships. Merrill's and Reigeluth et al.'s work described an ontology where the nodes can be high-level concepts, but each node contains facts, concepts, procedures, and principles. Escudero and Fuentes (2010) applied these concepts to intelligent tutoring and created learning objects that also have instructional objectives associated with them in an attempt to create a generic course generation authoring tool. That is, their tool creates a domain model with learning objects coupled with relationships between objects and learning objectives. The benefit in structuring the domain this way is that the domain becomes independent of an ITS (Escudero and Fuentes' goal) as the learning objects are self-contained units of instruction that can be exported to other systems.

Assessments can also be added to a learning object. Gibbons (2014) noted that task analyses can be used to create performance models as a part of the content layer. By including performance models or assessments in a learning object, learning objects can contain the information needed for the strategy layer to macro-adapt instruction, but also provide the information necessary for the other layers to present assessments (representation and message layers) as well as model student performance on a learning object (data management layer).

Escudero and Fuentes (2010) already presented a service that modularizes the domain model of a traditional ITS. Other services also exist that can extract prerequisite relationships of text-based learning objects (Gasparetti, De Medio, Limongelli, Sciarrone, & Temperini, 2018), generate assessment question from an ontology (Khodeir, Wanas, Darwish, & Hegazy, 2014;

Vinu & Sreenivasa, 2015; Žitko, Stankov, Rosić, & Grubišić, 2009), or design learning objects (Stamey & Saunders, 2005; Vassileva, 1995).

Summary. In summary the content layer covers much of the domain model. It is comprised of two sublayers: knowledge representation and learning objects. These two sublayers provide a starting point to conceptualize the basic learning unit of an ITS (learning objects) coupled with the structure needed to organize and communicate content (knowledge representation).

Data Management Layer

The data management layer defines how data is "gathered, remembered, analyzed, and how it is used beneficially" (Gibbons, 2014, p. 44). This layer most aligns with the traditional student model of a typical ITS. However, looking into the nuances of this layer as a service, we can start to see how the sublayers add a level of depth to the traditional student model in a way that promotes modularity.

The sublayers of data management are: data storage, the student model, the tutor model, and detectors. These sublayers represent what a traditional student model does, but it separates parts like the data storage from the student model so that a change in one sublayer can be made as independently as possible from another sublayer. Additionally, it allows additional functionality like the tutor model to be added to an ITS. This tutor model is not to be confused with the traditional tutor model of an ITS. Rather, this looks at modeling the ITS much like a student with the aim of helping the ITS be intelligent and learn how to be a better tutor.

Data storage. All the data collected by an ITS need to be stored long-term. These data can include information from student log data to ITS tutoring decisions to the structure of the content layer. The data storage sublayer holds all the information gathered by an ITS. Holstein et al. (2018) describe the benefit of keeping a central database (of the student model specifically) as supporting different use cases. For example, a central database is useful for providing current statistics on class-level analytics whereas a local copy of the student model is useful for making tutoring decisions about a single student.

Many services already exist that provide data storage. One example, LearnSphere (n.d.), exemplifies the opportunities afforded by having a separate sublayer for data storage. LearnSphere provides a service called DataShop that acts as a secure data storage of student interactions with educational software. Users can choose to import data or have data logged directly to DataShop from a course. DataShop provides a standard XML logging format so that data are collected in a uniform format. Additionally, DataShop has a suite of statistical tools to analyze the data. Users can also explore other datasets stored in DataShop that are shared publicly.

Student model. The student model serves as a subset of the overall data collected by an ITS representing a cognitive model of a student—a well-established practice in ITSs. The characteristics collected by an ITS vary from system to system in part due to the methods used for modeling a student. This variance in characteristics and methodologies contributes to the lack of reusability of student models. Additionally, Holstein et al. (2018) argued that innovative student modeling techniques rarely make it to live tutors or new settings. Aguirre, Uresti, and Boulay (2016) affirmed this in a review of student models in terms of their portability. While they found that some student models were reused in new settings, it seemed difficult to say whether this is possible without a deep understanding of both the model and the system. The researchers who designed the student model and system may be able to port a student model, but third-party researchers would find it very difficult to achieve similar success in porting. The key to being successful seems to be a field-wide standardization of student models and data, but Aguirre et al. called this a “faraway reality” (2016, p. 968).

Holstein et al. (2018) attempted to reach this faraway reality by extending the authoring tool CTAT to include the capability of a modularized student model. This tool is now called CT+A. Authors are now free to add variables to the CT+A student model and can craft tutor behavior to respond to these new variables. CT+A is a great step forward in creating a services-based ecosystem. However, it still falls short of a truly modularized system. In fact, Holstein et al. (2018) noted some changes would have required “a substantial re-architecting” to simplify adaptivity of the system. CTAT was not built to be extensible and will encounter new problems if additional parts were to be extended.

Tutor model. The tutor model, another subset of the data collected by an ITS, represents the cognitive model of a tutor. Some confusion may arise here between this tutor model sublayer and the tutor model traditionally found in ITS architecture. To clarify, the distinction between the two is best understood by recognizing that tutorial planning and execution belong to the strategy layer. Typically, the traditional tutor model has instructional strategies and tactics built in, thus making it seem like the tutor model is equivalent to the strategy layer. However, information about tutoring decisions needs to be recorded, hence the need for a tutor model as a sublayer in the data management layer. The purpose of recording this information is to allow the strategy layer to be agnostic in terms of instructional theories and strategies. Instead, it arbitrates instructional decisions. In order for those decisions to be informed, a history of tutorial decisions (the tutor model) given a particular student (the student model) allows the system to adapt instructional strategies based on the student rather than the strategies being fixed and built into the system.

An example may elucidate this overlap of terms. AutoTutor is an ITS that uses dialogue as its primary instructional strategy (D'Mello, Graesser, & King, 2010). That is, when teaching conceptual physics, AutoTutor will ask questions and guide students toward an ideal answer via a conversation between system and student. AutoTutor has many tactics it can use to provide that guidance. The strategy of dialogue is useful in certain contexts. What happens if that strategy is not working for a student? AutoTutor cannot abandon its strategy because it is built into the system. If AutoTutor were agnostic in terms of the strategies it could choose from, it would be free to choose a strategy that would work best given the context. How would AutoTutor know what strategy to switch to in this situation? If it had a tutor model, akin to the student model, that kept track of the system's performance, it could then make an informed decision based on previous experience with the student by choosing a particular strategy that historically has worked well with the student. This modeling process is considered the tutor model in this context.

Detectors. Gibbons' (2014) definition of the data management layer mentions how data is analyzed and used beneficially. The data storage, student model, and tutor model sublayers are lacking when it comes to these two functions and for good reason. Separating these two parts of the definition into a sublayer allows a system to be flexible and less opinionated in how data is analyzed and used.

Not only did Holstein et al. (2018) extend their student model to be open to the collection of new variables, they also allowed custom-built detectors to plug into their architecture (and even provided a library of compatible detectors) to analyze these new variables. These detectors' analysis of a student's data stream is used to suggest tutor behaviors. For example, a detector could be built to identify when a student is wheelspinning (i.e., a state where the student is stuck in a mastery loop, unable to progress due to lack of skill or knowledge). Once this behavior is identified, the detector could suggest any number of ways for the tutor to help the student out of wheelspinning, at which point the strategy layer would takeover to arbitrate tutoring decisions.

Similarly, detectors should be built for the tutor model. These detectors could be more evaluative in nature, describing the overall performance of the tutor. For example, a detector could analyze the tutor model and recognize that a particular instructional strategy has not been having the anticipated effect on any students. This could prompt the system to alert the authors that something may need fixing within their service.

The idea of detectors is not new in the ITS literature. However, they do seem to be built for a specific system. CT+A, as an example, allows for custom-built detectors, but the detectors are custom-built to the CT+A architecture and not built modularly. As such, these detectors are unlikely to be able to be used in other systems. The problem continues to be a lack of standardization in data management.

Summary. The data management layer expands the scope of data storage required of an ITS and, perhaps, explicitly defines aspects of data storage that went unnoted in previous ITSs. The four sublayers included here are: data storage, the student and tutor models, and detectors. Data storage refers to all data that needs to be stored by an ITS whereas the student and tutor models are specialized subsets of the data. The subsets allow data to be organized in a matter that is conducive for the detectors to analyze and inform the ITS of various things like a student's emotional state or productive tutoring paths the ITS could take.

Strategy Layer

When it comes to the actual intelligence of an ITS, the strategy layer stands out. Akin to the tutor model in the traditional architecture, the strategy layer makes tutoring decisions based on the stream of data being passed to it. It is central to a layers architecture, hence Gibbon's assertion that "all of the other layers represent an extension of the strategy layer" (p. 41, 2014). To review, the strategy layer's core function is to match goals between the teacher (or tutor in this case) and the student with an eventual shift of burden from a teacher-centric to student-centric practice.

A typical tutor model has been defined based on its functions: sequencing content, providing opportunities for practice, and providing feedback (Colby, 2017). A similar structure is being used to describe potential sublayers. The sublayers are micro-adaptation, macro-adaptation, and meta-adaptation. One notable departure from the literature with these sublayers is the capability of layers to adapt learning theories, instructional strategies, and tutoring tactics. Typically, these behaviors and philosophical underpinnings are baked into the system and cannot be readily changed. However, a layered approach allows adaptation on this level as well. With that in mind, each sublayer below describes two parallel processes passed from one sublayer to the next. The first process is a refinement of learning theories into instructional strategies into tutoring tactics. The second process is the translation of learning goals into a tailored course that is then executed and adapted in real-time.

Micro-adaptation. Micro-adaptation, also referred to as the inner loop (VanLehn, 2006), refers to in-course or problem-based adaptation. For example, if a student is working on a concept but demonstrates a lack of mastery for a prerequisite skill, the system can adapt and start teaching the prerequisite skill even though it may not have been part of the original plan. It can also choose from a range of tutoring tactics afforded by the instructional strategy like providing feedback to the student.

Macro-adaptation. Macro-adaptation, or what has been called the outer loop (VanLehn, 2006), encompasses the planning capability of an ITS. That is, it takes care of sequencing a course for a student. It structures the learning objects in such a way as to meet the goals of the tutor and the student. Along the way, the tutor needs to make decisions regarding the instructional strategies that will be used. For example, will the tutor take an apprenticeship approach or will it go for a more direct teaching method? This will depend on a lot of factors, but most importantly on the learning goals of both tutor and student as well as the strategies that are consistent with the current learning theory that governs the system. These factors lie with the meta-adaptation sublayer.

Meta-adaptation. Meta-adaptation is more recently discussed in ITS literature. Nye (2015) described it as being able to swap between different systems or services to take advantage of the unique properties of each. Following the same thought, this sublayer is responsible for identifying learner goals and the learning theory best suited for that goal. From there, a trickle-down effect occurs. Macro-adaptation sequences a course to meet that goal while choosing instructional strategies to accomplish that goal. Micro-adaptation then takes over, executes the plan, makes micro changes to the plan as needed, and chooses the tutoring tactics best suited for the job.

There have been some services identified in the literature that fill in parts of these sublayers (Nye et al., 2018; Vassileva, 1995). However, these services are detached from the capability of adapting theories, strategies, and tactics. For that reason, theoretical services will be described for this layer to better demonstrate the synergy of the sublayers, their relationships to other layers, and the resulting intelligence of the strategy layer.

Starting with the meta-adaptation layer, the first step is for the learning goal to be decided. The decision-maker here can be either the tutor or the student. Let us say the tutor decides that the student should learn the concept of single-digit multiplication based on a gap in skill of the student model. Additionally, the tutor decides that while the student has demonstrated mastery of single-digit multiplication, the student has not demonstrated fluency. Therefore, the tutor decides a behavioristic approach would be more appropriate to gain fluency for this concept based on predicted student outcomes from the data management layer.

From there, the macro-adaptation sublayer takes over and looks at the available learning objects for single-digit multiplication. Using a planning algorithm, it sequences instruction, examples, and problems for the student based on the content layer. Of the available instructional strategies for behaviorism, the tutor decides that the student would benefit best from immediate feedback to shape the correct behavior. Once that's completed, the micro-adaptation sublayer is responsible for executing the plan, making minor course corrections as necessary, and providing feedback to the student. In this case, feedback could come in many forms. It could be as simple as flag feedback (i.e., a simple demarcation of correct/incorrect) or more in-depth in the form of explanations.

The micro-adaptation sublayer needs to choose a tutoring tactic from a range of behaviors allowed by the instructional strategy. Let us say the student is highly motivated and learns quickly, so the micro-adaptation layer determines flag feedback is enough to help the student get the right answers. Therefore, the student works through the current sequence until the student demonstrates fluency of single-digit multiplication.

With this example, we see the two strategic processes simultaneously occurring throughout each sublayer and that is where the need for services is. On one hand, we need the learning theories, instructional strategies, and tutoring tactics to be operationalized. That is, turn them into an actionable format for an ITS. Isotani, Mizoguchi, Inaba, and Ikeda (2010) operate on the foundation that every learning theory has a common basis for explaining learning even though the how or why of learning occurs differs in each theory. They argue that the common basis is the idea of states or stages, and that learning occurs as students shift from one state to another. Using Rumelhart and Norman's (1978) theory of knowledge acquisition and Anderson's (1982) theory of skill development, they created an ontology of different states of skill and knowledge. They called this ontology the learner's growth model.

In short, this model could allow an ITS to determine where a student is at in terms of knowledge and skill, based on the student model. After that, the tutor could explore best-fit paths that lead to the goal state. For example, from one start state, a student could reach the goal state by direct instruction or a cognitive apprenticeship. Since the learner's growth model allows us to model the state changes of those instructional strategies, the macro-adaptation sublayer can make a decision regarding which strategy to use.

Caro, Josyula, and Jimenez (2015) also operationalized theories with an ontology, but not in terms of stage changes. Rather, they map the relationships between theories, strategies, and tactics with available resources and the student's learning style in order to represent the context of the learning environment. Their ontology provides a structure for operationalizing theories, strategies, and tactics in an actionable way.

The second process is the translation of learning goals into course sequences, which is then executed and adapted at a fine-grained level as needed. A mediator is required to handle all the sources of input (e.g., student data or teacher goals) to determine a learning goal. A planning algorithm is needed to determine the course sequence. Finally, an executor is used to implement the plan.

While this section describes the service horizontally, it may be useful to conceptualize these services vertically. That is, a service that encapsulates all three sublayers. Kay (2001) described interchangeable services that allow a student to choose a teacher or peer to work with while interacting with the ITS. For example, a student could choose to work with a constructivist teacher who has the three sublayers built in as a personality of sorts. Conceptualizing the strategy layer this way may make it easier for students and authors to interact with the strategy layer.

Summary. The strategy layer can be considered the intelligent part of an ITS. Within this layer, decisions are made that allow the ITS to adapt to a student on three levels: micro-adaptation, macro-adaptation, and meta-adaptation. These three levels represent the sublayers of the strategy layer and can be conceptualized either horizontally or vertically. The horizontal conception addresses each sublayer on its own—the creation of services that provide different forms of micro-adaptation as an example. The vertical conception addresses the relationship between each sublayer—the flow from meta- to macro- to micro-.

Representation Layer

The next three layers (representation, control, and message) comprise the interface of an ITS. Typically, these three layers are so integrated at the time of the instruction that it becomes difficult to tease apart what these layers look like independent of each other. In fact, ITS researchers seem to see the interface as an extension of the strategy layer and develop the interface with the strategy baked in (Colby, 2017). However, this mindset leads to too much focus on the representation layer of an ITS with little regard to the design of the message and control layers. This leads to a rigid interface design that does not support modularity. For that reason, each layer is treated separately to better understand the modularity of the interface even though in practice the interface is comprised of three inter-dependent layers. For the following sections, when referring to these three layers combined, the word interface will be used.

Chughtai, Zhang, and Craig (2015) made the observation that early ITS research focused on the importance of human-computer interaction (HCI), but that the research community shifted towards the educational aspects of ITS. Colby (2017) corroborated this finding in a literature review that found that while there are studies that explore the HCI of an ITS, those studies are few and there has not seemed to be a focus on user experience/user interface (UX/UI) research within ITS. That being said, the answer to the question above needs to be made within the context of the broader UX/UI research community. Unfortunately, a full review of that research is outside the scope of this paper. Instead, the separation of the interface into three layers may provide a way for ITS authors to develop interfaces that are not so enmeshed with the design of the system and strategy.

With the shift to web-based ITSs, the structure needed for modular representation already exists. For example, HTML provides the structure of an ITS's representation while the CSS applied to that web page affects how it looks. Additional technologies allow web pages to be loaded dynamically, which means control mechanisms can change based on instructional goals or that messages may adapt their display based on the expertise of a student. All this can happen with little disruption to the representational structure.

Representational services for the web already exist that make it easier to customize the display of an ITS. Take, for example, Bootstrap (Otto & Thornton, n.d.). Bootstrap is a library that allows for easy customization of the view and functionality of a web page. Once a web developer knows what features the library provides, it becomes much easier for them to create a web page. While Bootstrap works for web development, there are aspects of web-based ITSs that require more of an educational approach for a service to be useful.

Delphinium (n.d.) is an example of a service that not only provides different elements with a unified theme, but also provides an educational approach. Delphinium, a plug-in for the Canvas LMS, changes the look of the course to support gamification. It uses the data provided by Canvas to adjust the representation to fit a different instructional strategy than what Canvas typically provides. While Delphinium is rather opinionated in the instructional strategy being represented (i.e., self-regulated learning through gamification), it highlights the fact that representational services can be made that fit the needs of an ITS. All the data that Delphinium utilizes already exists within the Canvas course. A more general service like Bootstrap that can anticipate instructional elements could be used on a larger ITS scale. For example, a library with multiple displays for dashboards that present data on the student could provide messages about the student's past performance and future path. Or, perhaps a library that could adapt the display based on the current instructional goal. For example, if a skill-and-drill approach is required, there could be a representation that focuses on showing multiple problems and allows for rapid user input. Alternatively, if the current goal requires self-explanation via dialogue, the representation could display a pedagogical agent with natural language capabilities.

The above description works well for web-based ITS where the modular structure is already in place. However, the representation layer is dependent on how it is being accessed. For example, while a web page may have a mobile-friendly version, what if an ITS was built as an app for a phone or a tablet where internet connectivity were an issue? Some programming languages may be able to accommodate a modular interface better than others and this needs to be a consideration for development of services for the representation layer.

The representation layer refers to the part of an ITS that a student actually sees and interacts with. It is here that design decisions are made regarding the aesthetic qualities of an ITS. Tightly integrated with the representation layer are the control and message layers, which are discussed next.

Control Layer

There are two conversations when talking about the control layer. The first conversation is about control on an abstract level while the second is on a concrete level. Each conversation touches on the purpose of the control layer—how the student communicates to and performs an action with the system. Gibbons (2014) touched on the abstract conversation when he mentioned that the overall strategy of an instructional artifact should shift responsibility for learning from the system to the learner. Such a shift would have implications for the design of the control layer, meaning that the level of control a learner has needs to increase as expertise is gained. While a student being autonomous and responsible for his or her own learning is a desirable goal, the context of the instruction needs to be taken into account. For example, mastery learning, a popular paradigm in ITSs, cannot be utilized to its full extent as school years and traditional classroom structure limit the amount of time a student can spend on a unit in order to gain mastery. Similarly, the sequencing of content is often constrained by a teacher and the course schedule. These things need to be kept in mind when designing a system that has controls that can be adapted.

The second conversation, and what the remainder of this section addresses, is about the actual controls displayed in a system. Controls can broadly be defined as anything that allows the user to communicate with the system (e.g., input fields, command buttons, menu options). The importance of thinking of the controls as modular comes to light once we understand the relationship between the control and strategy layers. The strategy layer should dictate which controls are available to the student for the purpose of achieving an instructional goal under a certain pedagogy. However, not all controls need to be modular or are influenced by an instructional goal. The four sublayers below (domain controls, generic controls, instructional controls, and system controls) clear up the relationship between various controls and the strategy layer. These sublayers also help us understand the question of which parts of the interface should be adaptive or adaptable.

Before addressing potential services of this layer, it is important to make a distinction between the adaptive/adaptable nature of the interface (Jameson, 2008). *Adaptive* interfaces adapt its behavior to the users based on data about that user. *Adaptable* interfaces allow the user to adapt the interface to fit their preferences. Designers must ask themselves the question, "Which parts of the interface need to be adaptive and which parts need to be adaptable?"

Domain controls. Domain controls are specific to a domain. For example, math domains may require controls that allow a student to input complex formulas. A physics domain may make use of a simulated environment to demonstrate the effects of physics on simulated objects (Graesser, Jackson, Kim, & Olney, 2006), whereas programming domains require language-specific text-editors, debuggers and compilers. These controls generally should not need to be adaptive to a learner. While some controls may seem specific to a domain (e.g., a calculator), these controls can be considered as part of the next sublayer, generic controls.

Generic controls. Text editors, spreadsheets, calculators, buttons, inputs, hint requests, feedback, and pedagogical agents/chatbots are all examples of generic tools because they could be ubiquitous controls regardless of the system. Many ITSs can make use of these tools so there is no reason to duplicate efforts to create them across systems. A modular approach would enable the use and refinement of generic controls. These controls are also not usually adaptive to the learner. An example of a generic control is the Virtual Human Toolkit (VHTk; Brawner et al., 2016). This toolkit provides the functionality to model virtual humans, or pedagogical agents. It can handle speech processing, emotional modeling, gestures, etc. While this toolkit would be part of the message layer (a message would likely be sent through the pedagogical agent), it does allow the student to use speech as a mode of input.

Instructional controls. Instructional controls are those that reflect the current goal, strategy, and tactics of the strategy layer. For example, if the system is currently focused on social learning, learners will need tools that allow them to work together as a group. Kay (2001) provided an example of social learning tools that could become available for students to divide the task to group members, provide assistance to each other, track progress, etc. Aside from adapting to the strategy layer, instructional controls can also be adaptive to a student. Previous ITSs have implemented a math solver that completes part of a problem a student has already mastered so that the student can focus on the current skill at hand rather than slipping and getting the problem incorrect (Schulze et al., 2000).

System controls. System controls can also be considered ubiquitous like the general controls described above. Whereas general controls control the instruction, system controls are meant for controlling functions of the system. Examples include controls for preferences, window, or tool layouts similar to the way Adobe products provide customized “Developer,” “Designer,” or “Artist” layouts. These controls can allow the system to be adaptable—changed to meet the needs of a student. Additionally, some researchers have claimed that there is value in being able to negotiate the beliefs of a system and the beliefs of a learner. A practical example is that of having an open or negotiable student model. Kay (2001) said, “If a learner is expected to take responsibility for their own learning, it seems inconsistent to expect them to tolerate an incomprehensible, inscrutable system that manages their learning" (p. 112). While controls for negotiating with the system are valuable when it comes to incorrect or incomplete student models, additional benefits can be had as the control and responsibility for a student's learning shifts from the system to the student.

Summary. The control layer allows a student to communicate with a system, expressing their desire to submit an answer, seek help, or to interact in some other way. Four sublayers were discussed that demonstrate the various levels of control and to highlight the need to be adaptive vs. adaptable. The four sublayers are: domain controls, generic controls, instructional controls, and system controls. Domain controls are specific to a domain and are meant to interact specifically with the content from that domain (e.g., a physics simulation), whereas generic controls are suitable across multiple domains (e.g., a calculator). Neither typically require any level of adaptivity. Instructional controls are used to communicate the current goal, strategies, and tactics of the system like tools that facilitate social learning and are conducive to adaptivity. Finally, system controls allow the user to interface with the system as a whole and can bring in levels of adaptability.

Message Layer

The message layer answers the question, "What needs to be communicated from the system to the learner?" The answer depends on the layer that needs to be communicating. The content layer needs to be communicated somehow, the data management layer may need to communicate the student model, and the instructional goals need to be communicated to the learner from the strategy layer. A second question that needs to be answered, and is likely the source for services, is, "How does the system communicate the messages from the other layers?"

Perhaps that communication comes in the form of video, images, text, or even a pedagogical agent. How this starts to take shape may depend on the communication needs—macro-level communication (e.g., instructional goals) may be better suited through a pedagogical agent whereas micro-level communication (e.g., feedback) may be better served with text or graphics handled by the representation layer. The following sublayers for the message layer start to emerge from this understanding: strategic communication, content communication, and data management communication.

Strategic communication. This sublayer involves the communication of the three aspects of the strategy layer: instructional goal, pedagogical strategy, and tutoring tactics. A strategic communication service would take these different parts and find a way to communicate them to the student. For example, a system could communicate instructional goals to a student through a knowledge tree like Duolingo (n.d.) does for learning a language. A student can select an instructional goal in this case. Another method may be to communicate goals as badges or microcredentials. From there, a pedagogy needs to have a message mechanism that allows for the implementation of that pedagogy, such as natural language dialogue. Natural language dialogue requires a student and system to communicate with one another, but how? It could be through spoken input or text input, both of which require the message layer to communicate that appropriately. Lastly, the tutoring tactics need to be communicated. An example is a worked example. Trying to use spoken input to show a worked example may be ineffective. Instead, an image or text may be better suited to represent that message. These aspects of strategic communication need to be considered in the design of an ITS. The benefit of having a message layer separate from the strategy layer is that as the goals, strategies, and tactics change, so, too, can the message. Likewise, while the strategy may remain the same, the system might try several different messages to help a struggling learner. In this case, machine learning could be used to help the system learn, over time, which messages best fit the desired strategy for the chosen student model.

Content communication. This sublayer answers the question, “How will content be displayed?” Will images, videos, text, or simulations be used? The choice is constrained by what is made available to the learning object for the current content. For example, if a student wants to complete a simulation to learn the principles of chemistry, the data for the simulation need to be available and associated with the learning object. However, the data for the simulation are not enough. A simulation engine would be needed to take the data from the learning object and display it to the student. While the actual simulation engine may be part of the representation layer, the simulation engine needs to have an avenue set up for the system to communicate the simulation data of the current learning object—the message layer. By being cognizant of the need for a message layer to communicate with the representation layer, it allows the interface of the system to be designed in such a way to handle different formats of content displays. Not only that, but also more research can be done to look at how the message is communicated. For example, McLaren, DeLeeuw, and Mayer (2011) looked at how the tone of the message can affect a student’s learning. By using a more polite tone, they were able to facilitate better learning.

Data communication. Above, open learner models were discussed as part of system controls that could be designed for an ITS. The idea of communicating the student model is not a new one and has been implemented in ITSs previously (Aleven & Koedinger, 2002). While this type of data communication is already accepted practice within ITS, a layers perspective helps to conceptualize what other types of data communication might occur. Communicating the tutor model, for example, could open new avenues for the students to negotiate their learning. Rather than being able to negotiate only what a system believes the student knows, a student could now negotiate what they view as effective teaching practices, their responses to system changes, and how they feel about the UX/UI of the system. Opening up the system even more in this way can allow for a more natural tutoring relationship between the student and system. After all, students with human teachers can communicate whether or not a teaching strategy is effective, or if they feel like the teacher is making a correct assessment in what the students know.

Summary. Whereas the control layer allows the user to communicate with the system, the message layer is the system's communication channel with the user. Various communication needs exist for a system, and these needs comprise the sublayers for the message layer. Strategic communication refers to the communication of the strategy layer and its sublayers. Content communication is the communication of the content (i.e., as text, video, images, etc.). Data management communication informs the student of the data being kept (e.g., a learner model).

Media-Logic Layer

With so many separate layers and services, there needs to be some mechanism that brings them together. The media-logic layer is the glue that allows layers to communicate one with another. This layer also packages the layers and delivers them as a cohesive whole. The sublayers for media-logic are a communication sublayer, a delivery sublayer, and an authoring tool sublayer.

Communication. Being able to communicate effectively between modular components is not a new problem. Even as the call for more modular ITS architecture has been around for years, so, too, have attempts at finding a solution to this problem. In Chepegin, Aroyo, De Bra, and Houben (2003), the authors implemented a modular architecture for an adaptive, web-based system. Using their own components as well as third-party ones, the authors attempted to standardize the protocols for message exchange between components. In Trella, Carmona, and Conejo (2005), an open, service-based learning platform was created that also required communication between different services. One way they framed the problem was through a distributed software problem. For distributed software like web services, the communication is done through standard protocols that allow someone to access the service. These protocols describe what the service can do and how to access them. However, intelligent tutoring services are a special breed of service. It is not enough to be able to know how to access the service due to the interdependent nature of ITS. You must also understand how one service will impact another. For example, a learning object on economics may not have the requisite information for the strategy layer to choose a worked example tutoring approach. This introduces the idea that some services may constrain the use of another service if they are incompatible for tutoring.

While standardized languages have been adopted in related fields, issues have arisen that prevent ITS from adopting a standard. Too many systems are created in an academic silo where sharing and a cumulative history are not a concern. Rather, researchers are expected to use proprietary tools or languages to interface with these systems within a specific ITS paradigm (Nkambou et al., 2010). This observation leads Nkambou et al. (2010) to ask two questions. First, does this authoring bottleneck preserve the diversity of ITS or are standards needed for the survival of ITSs? Second, is the future of ITS a one-size-fits-all solution? More current research would suggest that the adoption of standards can do both—enable the survival of ITS while also allowing room for ITS diversity. This framework highlights those possibilities as well as provides a one-size-fits-all approach that allows sharing and creates building blocks for ITS development.

Perhaps one of the biggest advancements towards a standard is the Total Learning Architecture (TLA) that “offers learning systems developers an agreed-upon lingua franca” (Folsom-Kovarik & Raybourn, 2017, p. 4). Building upon xAPI, TLA allows for communication about additional topics on top of the learner experience. TLA can provide the communication channels between services, given their outputs are compatible with TLA. The reason for this is that TLA does not constrain how the services internally process data. Not only does this allow services to be developed independently of each other, but it also implies that the TLA can be swapped out for other communication services as technology develops. To help with those transitions, we can look at the efforts to create mediators that are capable of translating between different standards.

Mediators could be a possible service of the communication sublayer. These mediators allow a mapping to be created between different communication frameworks. For example, Kärger, Ullrich, and Melis (2016) highlighted a problem of learning object repositories. While it is useful to create learning object repositories, they are usually created in a proprietary fashion—with their own metadata and querying language (Kärger et al., 2016). Thus, many repositories exist, but it is not practical to try to pull learning objects from multiple repositories due to a lack of standards. Kärger et al.’s (2016) solution was to create a mediating architecture that translates queries to repositories into one each could understand. This is done using an ontology mapping and a query rewriting mechanism.

Delivery. When talking about the media-logic being the delivery method for all the services, it may be easy to conflate the interface with the delivery mechanism. For example, an LMS could be a good option for bringing together services much like the Canvas LMS does (Canvas, n.d.), and it may appear that Canvas is the delivery method. However, the true delivery service here is perhaps a computer accessing the web. Rather than constraining the idea of the delivery service to something web-based, it's useful to keep a more open mindset to accommodate the ever-changing landscape of technology. The delivery of an ITS will look different if it's a mobile or desktop app vs. a web-based solution. We can even start thinking of ITS being delivered in an augmented or virtual reality setting. These mediums have an impact not only on the interface, but also influence the types of strategies that can be implemented or the data that can be collected. While the decision seems to have been made that a web-based approach is the delivery method for the rising generation of ITS, this part of the media-logic needs to remain open to modularity.

Authoring tools. Authoring tools are a service that need to be addressed at some point when talking about ITS development. Authoring tools were designed in an effort to make ITS development faster and less reliant on programming knowledge. These tools are certainly useful, but they need to be reimagined to fit into a modular ecosystem. Currently authoring tools help developers create an ITS based on a specific brand of ITS. While some efforts have been made at exploring extensible authoring tools (Holstein et al., 2018), the majority of authoring tools are restricted to the instructional preferences of the authors. Instead, authoring tools need to be concerned not only with authoring, but also with putting modular pieces together. Because the media-logic layer is what brings the other layers together, it makes sense to include authoring tools as a function of this layer.

Due to this layer allowing the individual services to internally process data independently, the authoring tool will set up the communication channels between layers and choose an appropriate delivery method. For example, a teacher using the media-logic authoring tool will be able to select from an ecosystem of services to fulfill the role of each layer. The teacher could choose a history domain from the content layer, a mastery learning student model, coupled with a natural language dialogue model from the strategy layer, etc. After having selected these services through the media-logic authoring tool, the teacher can select an appropriate communication channel such as xAPI and a web-based delivery method. The authoring tool packages it all up and delivers the final product.

Summary. The media-logic layer brings together all the layers into a cohesive whole. Three sublayers facilitate that fusion: communication, delivery, and authoring tools. The communication sublayer sets up the channels for communication between layers through standard languages. The delivery sublayer refers to the vehicle that delivers tutoring (e.g., a human, the web). Finally, authoring tools, when redefined for a modular ecosystem, provide an interface that eases and expedites the process of creating ITSs by bringing together different services.

Relationships Between Layers

The layers for this framework were isolated and discussed above. A look at the layers as a whole ecosystem paints a clear picture of how this framework provides the necessary elements for transitioning into an ecosystem of services. The relationships described below follow the relationships pictured in Figure 2. Aside from one exception (the content layer), the relationships between the media-logic layer and the other layers will be discussed altogether at the end.

Starting with the content layer, Figure 2 shows the connections between two other layers: data management and strategy. The data management layer needs to store the content itself as well as the underlying structure. This information can be used by the data management layer to

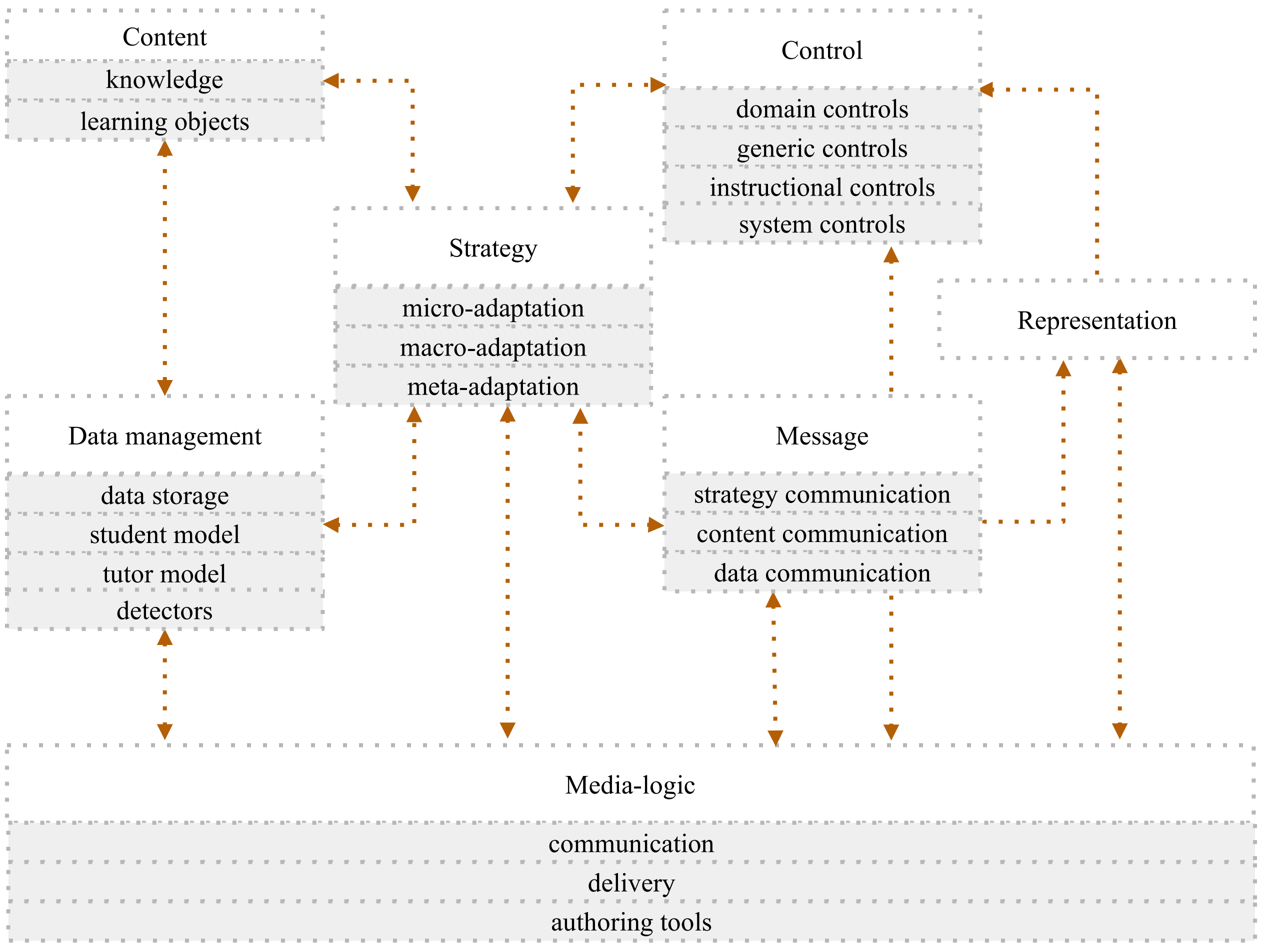

Figure 2. The layers, sublayers, and their relation to each other. Adapted from Gibbons (2014).

create a student model. While the content is stored within the data management layer, it must be considered a separate layer in order for modularity to exist. For example, learning objects may exist in an external repository or new advances to knowledge representation may require an overhaul of the structure of the content. The relationship to the strategy layer can best be illustrated when considering the learning objects available. If a learning object has no information stored for an essay assessment, the strategy layer cannot make the decision to assess a learner in that way. In other words, the content layer constrains the choices of the strategy layer due to the availability of material. A notable lack of a relationship between media-logic and content should be discussed. The media-logic engages with every other layer in this framework, so why the exception? Simply put, content should be independent of any system. An ontology of history can be used by any media-logic be it human or computer. Thus, media-logic interacts indirectly with the content through other layers.

Data management has a relationship with the strategy layer in addition to the content layer (which was already discussed). The most common relationship in existing ITS between data management and the strategy is to communicate the student model. However, the data management layer expands the information sent to the strategy layer by including information from the tutor model that can be used to create a self-improving system. The detectors sublayer analyzes the data and sends information to the strategy layer where the strategy layer can arbitrate which policy it should follow with regard to the current learning goal.

The strategy layer has connections between two additional layers: control and message. The strategy layer can determine which controls are available to a student allowing the student to communicate with the system in varying ways. It also communicates with the student through the message layer through methods like flag feedback or communicating the learning goal.

The message layer sends information to be represented by the representation layer. While this is typically conflated with the representation, separating the two allows a designer to recognize that the messages can be represented in multiple ways, sometimes simultaneously.

The control layer is also shown by the representation layer, but is separated for the same reasoning as the message layer—to allow for different forms of communication from the student.

The representation layer brings together the message and the control layers. Together, these three layers are the interface, and it is the only part of an ITS to have direct interaction with a student.

Finally, the media-logic's relationships to the other layers can be illustrated by exploring the human teacher as media-logic example given earlier and contrasting that with computer-based media-logic. Starting with data management, the media-logic can affect how accurate data collection is. A human can only store and process so much information and is error-prone. They may also find it difficult to express why they believe a student knows x but not y. A computer must be much more explicit and scalable. The strategy layer and media-logic have a relationship of constraints. A human is much better at facilitating a natural language dialogue as a tutoring strategy. If an ITS were to be accessed on a mobile device, natural language dialogue may be more difficult given the nature of a smaller keyboard and the potential lack of spoken input depending on the environment. The relationship between media-logic and control can be highlighted by looking at the hardware available in a computer setting. Perhaps a microphone is not available at all and so spoken input cannot be used, whereas a teacher may rely mostly on spoken input. As for the message layer, the media-logic determines how the message is delivered. For example, a teacher relies heavily on spoken communication to explain a concept, whereas a computer can use text, graphics, and simulations to express the same concept. Lastly, the relationship between media-logic and the representation is a simple concept to grasp. A teacher offers a very limited representation (themselves as a person), but can draw on other media-logic (e.g., books) to offer a more diverse representation. At that point, the book becomes the media-logic. A computer can do this as well, but perhaps more fluently.

Understanding the relationship between these layers allows an ITS designer using this framework to see how the ecosystem works as a whole. The sublayers presented here are just starting points. Additional sublayers can be added as needed based on the nature of the relationship between layers, but one must understand those relationships first. The above examples are merely that—examples constrained by current ITS practices. A more in-depth look at these relationships can only happen within the context of actual services interacting. For that to happen, ITS developers need to shift their focus from creating independent systems to creating interdependent intelligent tutoring services.

Conclusion

If ITSs are shifting to a services-based ecosystem as Nye (2015) predicts, a new framework will be needed to guide the design of intelligent tutoring services. A new framework can help shift the thinking of ITS authors from creating systems to creating services, while embracing the modularity inherent in a services-based economy. The current architectures used to develop ITSs are too rigid in their approach, as has been demonstrated by decades of research resulting in paradigmatic silos at the cost of a cumulative history. That is, a field where advances in research are isolated to a single system, unable to be adapted by others (Aguirre et al., 2016). This is not to devalue the research that has been done over the years. Indeed, that research history was necessary for the field to reach a point where cooperation, through a services-based economy, can lead the development of the next-generation of ITSs.

Using Gibbons' design layers (2014) provides a framework that guides the operationalization of the many services needed to design, develop, maintain, and ultimately proliferate effective ITSs. Built with modularity and cooperation in mind, a layers-based framework provides a flexible base from which we can explore the possibilities of an ecosystem of intelligent tutoring services.

References

Aguirre, B. V., Uresti, J. A. R., & Boulay, B. D. (2016). An analysis of student model portability. *International Journal of Artificial Intelligence in Education*, *26*(3), 932–974. doi: 10.1007/s40593-016-0113-0

Aleven, V. A., & Koedinger, K. R. (2002). An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor. *Cognitive Science, 26*(2), 147-179.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89*(4), 369–406.

Arroyo, I., Mehranian, H., & Woolf, B. P. (2010). Effort-based tutoring: An empirical approach to intelligent tutoring. In R. Baker, A. Merceron, & P. Pavlik (Eds.), *Educational Data Mining 2010* (pp. 1-10). Pittsburgh, PA: Carnegie Mellon University

Baker, R. S. (2016). Stupid tutoring systems, intelligent humans. *International Journal of Artificial Intelligence in Education, 26*(2), 600-614. doi:10.1007/s40593-016-0105-0

Brawner, K., Goodwin, G., & Sottilare, R. (2016). Agent-based practices for an intelligent tutoring system architecture. *Lecture Notes in Computer Science Foundations of Augmented Cognition: Neuroergonomics and Operational Neuroscience*, *9744,* 3–12. doi: 10.1007/978-3-319-39952-2_1

Brooks, C., Winter, M., Greer, J., & McCalla, G. (2004). The massive user modelling system (MUMS). *Intelligent Tutoring Systems Lecture Notes in Computer Science, 3220,* 635-645. doi:10.1007/978-3-540-30139-4_60

Brusilovsky, P., Sosnovsky, S., & Shcherbinina, O. (2005). User modeling in a distributed e-learning architecture. *User Modeling 2005 Lecture Notes in Computer Science, 3538,* 387-391. doi:10.1007/11527886_50

Canvas (n.d.). Retrieved from https://www.instructure.com/canvas/.

Caro, M. F., Josyula, D. P., & Jimenez, J. A. (2015). Multi-level pedagogical model for the personalization of pedagogical strategies in intelligent tutoring systems. *Dyna*, *82*(194), 185–193. doi: 10.15446/dyna.v82n194.49279

Chepegin, V., Aroyo, L., De Bra, P., & Houben, G. J. P. M. (2003). CHIME: Service-oriented framework for adaptive web-based systems. *Conferentie Informatiewetenschap.* Retrieved from http://wwwis.win.tue.nl/infwet03/proceedings/3/

Chughtai, R., Zhang, S., & Craig, S. D. (2015). Usability evaluation of intelligent tutoring system. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, *59*(1), 367–371. doi: 10.1177/1541931215591076

Colby, B. R. (2017). *A comprehensive literature review of intelligent tutoring systems from 1995-2015* (Master's Thesis) Retrieved from ScholarsArchive. (7239)

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the craft of reading. *Knowing, learning and instruction: essays in honour of Robert Glaser.* Hillside, NJ: Erlbaum.

D'Mello, S., Graesser, A., & King, B. (2010). Toward spoken human-computer tutorial dialogues. *Human-Computer Interaction, 25*(4), 289-323. doi:10.1080/07370024.2010.499850

Delphinium. (n.d.). Retrieved from http://delphinium.uvu.edu/

Duolingo. (n.d.). Retrieved from http://duolingo.com/

Escudero, H., & Fuentes, R. (2010). Exchanging courses between different intelligent tutoring systems: A generic course generation authoring tool. *Knowledge-Based Systems, 23*(8), 864-874. doi:10.1016/j.knosys.2010.05.011

Folsom-Kovarik, J. T., & Raybourn, E. M. (2017). Total Learning Architecture (TLA) enables next-generation learning via meta-adaptation. *Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC).* Arlington, VA: National Training and Simulation Association.

Gasparetti, F., Medio, C. D., Limongelli, C., Sciarrone, F., & Temperini, M. (2018). Prerequisites between learning objects: Automatic extraction based on a machine learning approach. *Telematics and Informatics, 35*(3), 595-610. doi:10.1016/j.tele.2017.05.007

Ghorbel, F., Ellouze, N., Métais, E., Hamdi, F., Gargouri, F., & Herradi, N. (2016). MEMO GRAPH: An ontology visualization tool for everyone. *Procedia Computer Science, 96*, 265-274. doi:10.1016/j.procs.2016.08.139

Gibbons, A. S. (2014). *An architectural approach to instructional design*. New York, NY: Routledge.

Graesser, A. C., Jackson, G. T., Kim, H. J. J., & Olney, A. (2006). AutoTutor 3-D simulations: Analyzing users' actions and learning trends. In *Proceedings of the Cognitive Science Society* (pp. 1557-1562). Mahwah, NJ: Erlbaum.

Holstein, K., Yu, Z., Sewall, J., Popescu, O., McLaren, B. M., & Aleven, V. (2018). Opening up an intelligent tutoring system development environment for extensible student modeling. *Lecture Notes in Computer Science Artificial Intelligence in Education*, *10947,* 169–183. doi: 10.1007/978-3-319-93843-1_13

Ismail, K. N., Ahmadon, F., & Shahbudin, F. E. (2018). Instructional material development using ontology learning. *Social and Management Research Journal, 15*(2), 35-49. doi:10.24191/smrj.v15i2.4969

Isotani, S., Mizoguchi, R., Inaba, A., & Ikeda, M. (2010). The foundations of a theory-aware authoring tool for CSCL design. *Computers & Education*, *54*(4), 809–834. doi: 10.1016/j.compedu.2009.09.010

Jameson, A. (2008). Adaptive interfaces and agents. In J.A. Jacko (Ed.), *The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications* (pp. 305-330). Boca Raton, FL: CRC Press.

Jarvis, M. P. (2004). *Applying machine learning techniques to rule generation in intelligent tutoring systems* (Unpublished master's thesis). Worcester Polytechnic Institute, MA.

Kärger, P., Ullrich, C., & Melis, E. (2006). Integrating learning object repositories using a mediator architecture. *Innovative Approaches for Learning and Knowledge Sharing Lecture Notes in Computer Science*, 185–197. doi: 10.1007/11876663_16

Kay, J. (2001). Learner control. *User Modeling and User-Adapted Interaction, 11*(1-2), 111-127.

Khodeir, N., Wanas, N., Darwish, N., & Hegazy, N. (2014). Bayesian based adaptive question generation technique. *Journal of Electrical Systems and Information Technology, 1*(1), 10-16. doi:10.1016/j.jesit.2014.03.007

LearnSphere (n.d.). Retrieved from http://learnsphere.org/.

McLaren, B. M., DeLeeuw, K. E., & Mayer, R. E. (2011). A politeness effect in learning with web-based intelligent tutors. *International Journal of Human-Computer Studies*, *69*(1-2), 70-79.

Merrill, M. D. (1983). Component display theory. In C. M. Reigeluth (Ed.), *Instructional-design theories and models: An overview of their current status* (pp. 292-333). Hillsdale, NJ: Erlbaum

Moundridou, M., & Virvou, M. (2003). Analysis and design of a web-based authoring tool generating intelligent tutoring systems. *Computers & Education, 40*(2), 157-181. doi:10.1016/s0360-1315(02)00119-7

Nkambou, R., Bourdeau, J., & Psyché, V. (2010). Building intelligent tutoring systems: An overview. In R. Nkambou, R. Mizoguchi, & J. Bourdeau (Eds.), *Advances in intelligent tutoring systems* (pp. 361-375). Berlin, Germany: Springer.

Nye, B. D. (2015). AIED is splitting up (into services) and the next generation will be all right. *AIED Workshops, 1432*(4), 62-71.

Nye, B. D., Pavlik, P. I., Windsor, A., Olney, A. M., Hajeer, M., & Hu, X. (2018). SKOPE-IT (Shareable Knowledge Objects as Portable Intelligent Tutors): Overlaying natural language tutoring on an adaptive learning system for mathematics. *International Journal of STEM Education*, *5*(12). doi: 10.1186/s40594-018-0109-4

Otto, M., & Thornton, J. (n.d.). Bootstrap. Retrieved from https://getbootstrap.com/.

Protege. (n.d.). A free, open-source ontology editor and framework for building intelligent systems. Retrieved from https://protege.stanford.edu/.

Reigeluth, C. M., Merrill, M. D., & Bunderson, C. V. (1978). The structure of subject matter content and its instructional design implications. *Instructional Science, 7*(2), 107-126. doi:10.1007/bf00121929

Roll, I., Aleven, V., McLaren, B. M., & Koedinger, K. R. (2011). Improving students' help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction*, *21*(2), 267-280.

Rumelhart, D. & Norman, D. (1978). Accretion, tuning and restructuring: Three modes of learning. In J.W. Cotton, & R. Klatzky (Eds.), *Semantic factors in cognition.* Hillsdale, NJ: Erlbaum.

Schulze, K. G., Shelby, R. N., Treacy, D. J., Wintersgill, M. C., VanLehn, K., & Gertner, A. (2000). Andes: A coached learning environment for classical Newtonian physics. *The Journal of Electronic Publishing*, *1*(6). http://dx.doi.org/10.3998/3336451.0006.110

Stamey, J., & Saunders, B. (2005). Designing intelligent learning objects. *Fifth IEEE International Conference on Advanced Learning Technologies (ICALT05)*, *2005,* 323-325. doi:10.1109/icalt.2005.111

Suraweera, P., Mitrovic, A., & Martin, B. (2005). A knowledge acquisition system for constraint-based intelligent tutoring systems. *12th International Conference on Artificial Intelligence in Education (AIED 2005)* (pp. 638-645). Amsterdam, the Netherlands: IOS Press. http://hdl.handle.net/10092/348

Trella, M., Carmona, C., & Conejo, R. (2005). MEDEA: An open service-based learning platform for developing intelligent educational systems for the web. *Workshop on adaptive systems for web-based education: Tools and reusability (AIED'05)* (pp. 27-34). Amsterdam, the Netherlands: IOS Press

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education, 16*(3), 227-265.

Vanlehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist, 46*(4), 197-221. doi:10.1080/00461520.2011.611369

Vassileva, J. (1990). An architecture and methodology for creating a domain-independent, plan-based intelligent tutoring system. *Educational and Training Technology International*, *27*(4), 386–397. doi: 10.1080/1355800900270405

Vassileva, J. (1995). Dynamic courseware generation: At the cross point of CAL, ITS, and authoring. In *Proceedings ICCE'95 - International Conference on Computers in Education* (pp. 290–297). Singapore, Taiwan: International Conference on Computers in Education (ICCE).

Vinu, E.V., & Sreenivasa, Kumar P. (2015). A novel approach to generate MCQs from domain ontology: Considering DL semantics and open-world assumption. *Journal of Web Semantics, 34*(C), 40-54. doi:10.1016/j.websem.2015.05.005

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge*. Los Altos, CA: M. Kaufmann.

Žitko, B., Stankov, S., Rosić, M., & Grubišić, A. (2009). Dynamic test generation over ontology-based knowledge representation in authoring shell. *Expert Systems with Applications, 36*(4), 8185-8196. doi:10.1016/j.eswa.2008.10.028

ARTICLE II

Proof-of-Concept: An Intelligent Tutoring Service for the Content Layer

Brice R. Colby

Peter Rich

Department of Instructional Psychology and Technology

Brigham Young University

150 MCKB – BYU

Provo, UT 84602

Abstract

This article provides a proof-of-concept for the development of two intelligent tutoring services based on the layers theory of instructional design (Colby & Rich, 2020; Gibbons, 2014). The first service, key concept extraction, uses a machine-learning approach to extract potential key concepts. The second, prerequisite relationship extraction, attempts to determine if prerequisite relationships exists between concepts. These services are applied to nine textbooks: Excel, statistics, biology, accounting, history, calculus, precalculus, physics, and chemistry. While attempts at automating these tasks have been successful previously, this paper relies as much as possible on the structure and content of the textbooks rather than external sources. Promising results were found for key concept extraction, demonstrating that some algorithms produce better recall while others produce better precision across the board. However, results for prerequisite relationship extraction show this to be a complicated task that may require the use of external sources to achieve meaningful results.

Keywords: intelligent tutoring systems, intelligent tutoring services, layers theory, content layer

Proof-of-Concept: An Intelligent Tutoring Service for the Content Layer

Intelligent tutoring systems (ITSs) show a lot of promise academically. Some systems have produced learning gains nearly equivalent to that of a human tutor (VanLehn, 2011), yet few systems see widespread use. Design costs, resources, and issues with scalability contribute to the lack of proliferation. These and other issues could be resolved by switching to a services-based architecture for ITSs. A services-based architecture is an approach where modularity is at the forefront of the design process. Independent services are combined together to comprise the end product.

We applied Gibbons' (2014) layers theory for instructional design to ITSs. The result was seven layers that represent the architecture of an ITS, but from the perspective of a services-based architecture. Each layer has sublayers that provide direction for the creation of services. The purpose of this study is to test the feasibility of a services-based approach in the design of intelligent tutoring services specifically relating to the knowledge representation sublayer of the content layer (Colby & Rich, 2020).

Ontologies are one option that have been used to develop the domain model of an ITS. These ontologies are a relational knowledge map of the domain to be taught. While ontologies are generally formalized in an ontological language, they can be visualized in a way that resembles a typical concept map. Ontologies can be used to inform tutoring decisions, sequence instruction, and student modeling. For example, assume a student is attempting to learn skill C. The student performs poorly. The ITS could look at prerequisite skills A and B and determine that the student needs more practice with skill B before the student can confidently master skill C. The tutor would then start assigning problems based on skill B. Knowing and understanding the relationships between knowledge components and skills allows the tutor to adapt to the student's performance, a necessary feature for a system that claims to be intelligent.

Domain models can be labor-intensive to create as the process typically requires the authors of the ITS to team up with subject-matter experts to determine the scope, content, and relationships in an ontology (Heffernan et al., 2006). While labor-intensive, creating a domain model is also one of the essential first steps to creating any ITS as it provides the backbone on which many other services might rely.

The services described in this paper uses natural language processing techniques on open textbooks to perform two tasks: (a) automatically extract key phrases from a textbook, and (b) automatically detect prerequisite relationships between concepts, if any exist. While these tasks are automatic, the services do not automate the process of creating a domain model. Rather, the services provide a rough draft of a model that is subject to the author's discretion so that the model works for the context. This fundamentally changes the process from creating an ontology from scratch, which can take days for experts to create, to refining an automated product. Finally, this service provides a prototype of modularized services that can be used to start creating the content layer.

Literature Review

The argument for a services-based architecture in ITS design is not a new concept. In the early 2000s, researchers were already suggesting ITSs be broken up into services (Brooks, Winter, Greer, & McCalla, 2004; Brusilovsky, Sosnovsky, & Shcherbinina, 2005). More recently, Nye (2015) suggested that ITSs are at a point where a services-based ecosystem is necessary and points to other educational software that uses a similar approach. However, Nye did not go so far as to suggest a framework for developing intelligent tutoring services.

Layers Theory

Instead, we rely on the field of instructional design for a framework: Gibbons' (2014) layered design approach fits the given problem well. This approach emphasizes the need for modularity in design as each layer of a design ages at a different rate. Gibbons outlined seven layers that he argued were functionally present in any instructional artifact. These layers are: content, control, data management, media-logic, message, representation, and strategy. Gibbons clarified that additional layers and sublayers can be created based on the need of the instructional artifact. This layered approach could provide a useful framework as ITSs transition from systems to services.

We applied these layers to an ITS context. Each layer has sublayers that provide additional focus for potential services. This article focuses on the creation of a service related to the content layer, specifically the sublayer knowledge representation (Colby & Rich, 2020). The content layer is "concerned with the nature and structure of the knowledge to be learned and its capture in a form that can be used during design and delivery of instruction" (Gibbons, 2014, p. 43). When it comes to ITSs, the content layer is most associated with the domain model of a traditional architecture. Many methodologies already exist that allow some representation of the domain, some even (semi-)automatically (Jarvis, 2004; Ismail, Ahmadon, & Shahbudin, 2018; Suraweera, Mitrovic, & Martin, 2005). This paper explores an automated attempt at extracting key concepts (KCs) and prerequisite relationships (PRs) from existing content. These two tasks make up the building blocks of a domain model, especially with regards to knowledge representation. These are also the beginning steps of ontology learning, an essential first step in creating an ITS.

Ontology Learning

Asim, Wasim, Khan, Mahmood, and Abbasi (2018) provided a good survey of ontology learning. Ontology learning describes the process of constructing a domain model from text. They summarized their survey in three points. First, ontology learning systems rarely started from scratch. Instead, the systems would rely on seed words or an existing ontology as a starting point. Second, natural language processing shows promising results for extracting concepts from various sources. However, their third point highlighted the difficulties of extracting relationships between concepts, calling it a major challenge for the field.

Asim et al. (2018) categorized ontology learning techniques into a set of three classes: linguistic, statistical, and logical. Linguistic characteristics are based on the characteristics of a language and are mostly used for preprocessing text. However, linguistic techniques are still used for concept and relation extraction. An example is tagging the part of speech for words in a sentence. Statistical techniques rely on the statistics of the text without consideration for underlying semantics. A popular statistical technique is the TF-IDF (term frequency-inverse document frequency) metric that measures how important a word is to a document while taking into account words that occur very frequently by penalizing them. Lastly, logical techniques are used to create axioms, or rules, for the domain. These rules define the logic of the domain. These approaches are used for creating higher levels of an ontology beyond that of what is presented in this paper.

Hasan and Ng (2014) pointed out another class to be aware of—structural characteristics. While the linguistics of a text may hint at a concept, a structured document could also provide clues. Hasan and Ng gave an example with scientific papers. Most of the paper's keywords should appear in the abstract and introduction. If a corpus of documents has similar structures, that structure can be exploited. A table of contents, sections of text, and even glossaries can be used to aid in the development of models that extract KCs and PRs from a text. Three out of the four classes described above (linguistic, statistical, and structural) were used in the creation of the features for key concept extraction and prerequisite relationship extraction.

Key Concept Extraction

For key concepts, Parameswaran, Garcia-Molina, and Rajaraman (2010) defined a concept to be

> a k-gram that represents a real or imaginary entity, event or idea that many users may be interested in (i.e., is popular), and does not contain any extraneous words such that including them would identify the same entity (i.e., is concise) (p. 2).

This definition of concepts has been used in this paper when identifying candidate concepts. Parameswaran et al. (2010) assumed that all Wikipedia article titles count as concepts and used that assumption to set the k-gram limit to four (i.e., a concept consists of up to four words). Their reasoning behind this was based on a count of words in Wikipedia article titles. They found very few concepts had more than four words and, as such, four-word k-grams provide a reasonable cut-off point. Hereafter, k-grams are referred to as n-grams.

Asim et al. (2018) stated that KC extraction has seen promising results in the literature. These results suggest that this task is more of a solved problem than prerequisite relationship extraction. For that reason, the PR literature will be explored in more depth than the KC extraction section. However, additional information and research is presented in the Methods section as the features for KC extraction are described.

Prerequisite Relationships Extraction

Ontologies deal with multiple relationships, entities, and structure beyond that of PRs and KCs (Asim et al., 2018). These additional elements may be more salient based on the domain and may become more useful to represent as an ecosystem of services is developed; however, in this paper, we focus on identifying PRs only. Prerequisite relations can be valuable to other layers (e.g., when modeling student knowledge or sequencing instruction) whereas other relationships may best be left to the learning objects sublayer within the content layer (Colby & Rich, 2020). Learning objects could benefit from having an explicit relationship like *is-a* or *part-of* with other learning objects. Other works on relationships extraction have taken a similar approach of focusing solely on PRs in order to begin creating an ontology (Alzetta et al., 2019; De Medio, Gasparetti, Limongelli, Sciarrone, & Temperini, 2016; Gasparetti, De Medio, Limongelli, Sciarrone, and Temperini, 2018; Liang, Wu, Huang, & Giles, 2015; Wang & Liu, 2016).

For example, Miaschi, Alzetta, Cardillo, and Dell'Orletta (2019) proposed using linguistic features extracted from textual resources exclusively in order to infer relationships between concepts. Miaschi et al. relied on two sets of linguistic features: lexical features and global features. The lexical features were for single concepts while global features were for concept pairs. These features were extracted from Wikipedia pages where the concepts existed. The features were fed into several models to test the effectiveness of the models in predicting PRs. Generally, they found that their model worked well for in-domain experiments, but the models experienced a drop in performance when the models attempted to train across domains. Their conclusion was that lexical features work well within a domain, but not well across domains where the global features performed better.

Adorni, Alzetta, Koceva, Passalacqua, and Torre (2019) considered adding a temporal component to the list of features. The temporal feature is called burst intervals (Kleinberg, 2002). The underlying assumption is that a key concept would become relevant along a period of time (i.e., the frequency of the word may see a sudden spike). With regards to the text, time is measured by the linear progression of the text. Adorni et al.'s goal in introducing burst intervals was to capture the most relevant context for a given concept. Additionally, relationships between concepts could be assumed by the temporal ordering of concepts within a text. This highlights the assumption that if a prerequisite relationship exists, the co-occurrence of two concepts would be likely, although co-occurrence alone is not sufficient evidence of a prerequisite relationship.

Liang et al. (2015) introduced a single metric to measure prerequisite relationships. This metric, called RefD (reference distance), measures how differently two concepts refer to each other. Relying on Wikipedia entries for a concept, RefD measures how many times concept B is referenced in concept A and vice versa. Liang et al. noted some issues with using Wikipedia as a filter, though. First, the coverage for different domains could vary significantly. Second, the quality of labels could vary due to the crowdsourcing nature of Wikipedia. Regardless of these difficulties, their results showed that RefD outperformed existing baselines when predicting a PR.

Methods

The purpose of this study was to explore the design of intelligent tutoring services based on a framework that emphasizes modularity. Specifically, it is the first step in answering Nye's (2015) call of moving towards a services-based ecosystem. As such, the following are our research questions (RQs):

RQ1. In what ways can a services-based architecture inform the design of intelligent tutoring services?

RQ2. How do we validate the creation of a content-layer service?

RQ3. Is the creation of a content-layer service useful? For example, does a content-layer service expedite the process of creating an ITS considering the prohibitive design costs that typically prevent ITSs going to scale?

We answered RQ1 qualitatively. A series of lessons learned throughout the design process will be proffered as a way of showing the design-thinking process while using Gibbons' (2014) layers framework. We answered RQ2 quantitatively. We report standard metrics that measured the effectiveness of the models. We answered RQ3 both quantitatively as well as qualitatively. The time it takes to use the service is measured and reported as a benchmark for how quickly that process can be completed. We then discuss the output of the model and its utility in a services-based ecosystem.

Service Workflow

The workflow (see Figure 1) for this project involves two tasks: key concept extraction (KCE) and prerequisite relationship extraction (PRE). The underlying textbook material for both tasks comes from two places: OpenStax (n.d.) and an online Excel course provided through the platform MyEducator (n.d.). OpenStax was chosen for its open nature and the variety of subjects available. The Excel textbook was chosen as an outside sample that has a different format/structure than OpenStax.

For KCE, the workflow involved several steps (Figure 2). First, the KCs were extracted from the web where both OpenStax and the Excel textbook had a glossary of KCs. This was important for training a supervised model as these predetermined KCs provided the supervision for the models.

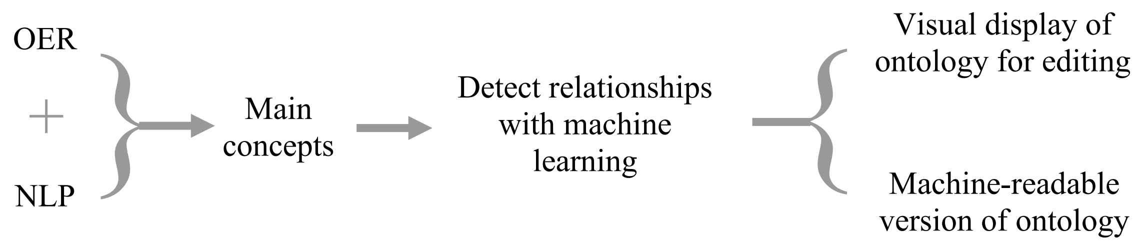
OER
+
NLP
Main concepts
Detect relationships with machine learning
Visual display of ontology for editing
Machine-readable version of ontology

Figure 1. A graphical representation of the service workflow. OER and NLP tools are used to extract concepts, detect relationships, and provide a human- and machine-readable output.

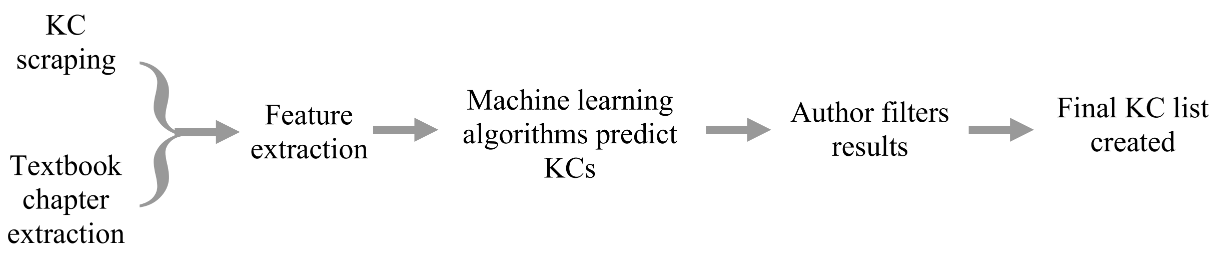

Figure 2. A graphical representation of the KCE workflow. KCs are scraped if available for supervision. Otherwise, the textbook's information is extracted to produce the features for machine learning algorithms to produce a final set of KCs.

Second, the PDFs of the textbooks were parsed to provide some structural information (i.e., chapter (sub-)headings and the table of contents). Additional features were extracted to provide the data needed for later processes. These features included chapter start and end pages, the chapter text tokenized by paragraph, and the n-grams for the chapter text. The n-gram word limit was set at four in accordance with Parameswaran et al.'s (2010) findings. Once the required information was gathered from the textbook, the features for the machine learning models were extracted. Fifteen features were extracted as part of this process.

TF-IDF: This feature served two purposes. First, to measure the overall relevance of an n-gram in a text. Second, to act as a filter. Before the TF-IDF score was calculated, the n-gram frequency served as a threshold for OpenStax. The n-gram needed to appear at least four times. For Excel, it was lowered to at least one appearance. A second filter, Wikipedia titles, was also used similar to Parameswaran et al.'s (2010) work. Assumption: The relevance of an n-gram would represent a candidate KC.

parDist: The distance between the start of the paragraph and the first occurrence of the n-gram measured in terms of percentage from the beginning. Assumption: KCs are introduced towards the beginning of a paragraph.

sentDist: The distance between the start of the sentence and the first occurrence of the n-gram measured in terms of percentage from the beginning. Assumption: KCs are introduced towards the beginning of a sentence.

spread: The distance between the first and last occurrence of an n-gram in a paragraph. Assumption: KCs would be the subject of a paragraph and would have a greater spread than non-KCs.

titleSim: A similarity metric based on the spaCy python library (n.d.). Measures the similarity between the n-gram and (sub-)headers in a chapter. Assumption: KCs are more similar to the (sub-)headings than non-KCs.

parts-of-speech: Nine features that measure the part-of-speech distribution for an n-gram. The parts-of-speech tracked are: *adjective, adverb, determiner, noun, number, proper noun, punctuation, verb,* and *other*. Assumption: KCs are normally comprised of certain parts-of-speech while others rarely, if ever, show up.

nounChunk: Using spaCy's noun chunk tagger, determines if the n-gram is a noun chunk. Assumption: KCs are likelier to be noun chunks.

One metric, RefD (Liang et al., 2015) was not included even though it has some predictive power. RefD relies on Wikipedia to calculate the reference distance. One goal of the current research was to use the textbook as much as possible without external sources. Two deviations from that goal occurred when using a list of Wikipedia titles as a filter for concepts and WordNet (n.d.) for supervising PRs. RefD relies on Wikipedia data in a manner that places less of the burden on the textbook. For that reason, it was not considered as a feature.

For PRE, two variables comprise the workflow (Figure 3). The first is the same chapter info that was extracted for KCE. The second is a list of KCs for the chapters. The workflow has four steps. First, a modified form of burst intervals is taken. Rather than using burst intervals to detect the relevance of an n-gram, the intervals are used to detect PRs. This echoes the assumption that co-occurrence of two words may indicate a relationship, but is not a sufficient indicator of one (Adorni et al., 2019). A window is detected for each KC. This window returns the sentence index of the chapter text where the KC occurs. A second window is detected, but is the inverse of the first window—each sentence index is given a value of all the KCs in that sentence. These two steps are cross-referenced in the third step to determine the concept pairs for detecting a prerequisite relationship. Concept pairs are considered if two concepts appear together in the temporal flow of the textbook. The fourth step is to extract the features for predicting a PR. Ten features are calculated based on other successful models (De Medio et al., 2016; Miaschi et al., 2019). A similar naming convention is used as well where At and Bt represent the KCs and A, B represent the windows of text for each concept (Miaschi et al., 2019).

inTxtBtA: Does Bt appear in the text of A? Assumption: If A reference Bt, PR may exist.
inTxtAtB: Does At appear in the text of B? Assumption: If B references At, PR may exist.
firstLineBtAt: Does Bt appear in the WordNet (n.d.) definition of At? Assumption: If Bt is used to define At, PR may exist.
firstLineAtBt: Does At appear in the WordNet (n.d.) definition of Bt? Assumption: If At is used to define BT, PR may exist.
inTitleAtBt: Is At part of Bt? Assumption: If At is part of Bt, then Bt may be a more general concept and PR of At.
inTitleBtAt: Is Bt part of At? Assumption: If Bt is part of At, then At may be a more general concept and PR of Bt.
simAB: Uses spaCy (n.d.) to calculate the similarity between A and B. Assumption: If the words used in A and B are similar, then the KCs are likely related to each other.
sharedNs: The number of shared nouns between A and B divided by the total number of nouns in A and B. Assumption: If At and Bt use the same nouns in a given context, they are likely related.
uniqueNounsA: The number of nouns unique to A divided by the total number of nouns in A and B. Assumption: At and Bt may not be related if the number of unique nouns is high between the two.
uniqueNounsB: The number of nouns unique to B divided by the total number of nouns in A and B. Assumption: At and Bt may not be related if the number of unique nouns is high between the two.

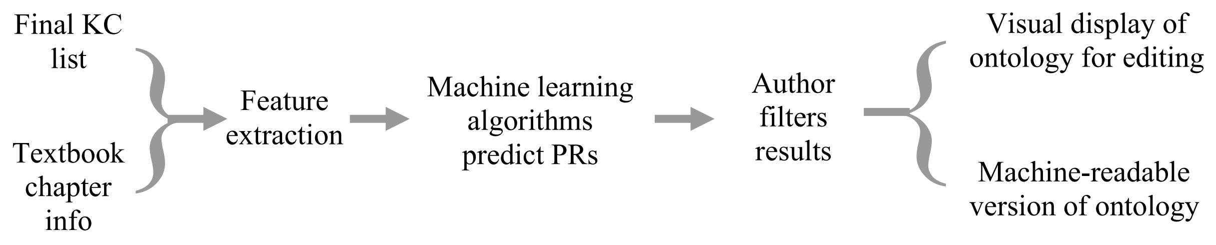
Final KC list
Textbook chapter info
Feature extraction
Machine learning algorithms predict PRs
Author filters results
Visual display of ontology for editing
Machine-readable version of ontology

Figure 3. A graphical representation of PRE workflow. Features are extracted from a KC list and the textbook. Algorithms predict PRs for an author to filter with the end result being an ontology.

Evaluation

Asim et al. (2018) laid out several evaluation techniques in their survey on ontology learning. While many techniques have been developed and proposed over the years, Asim et al. described four broad categories they thought encompassed the various techniques: golden standard-based evaluation, application-based evaluation, data-driven evaluation, and human-based evaluation. The evaluation for RQ2 is primarily human-based.

KCE is evaluated using a supervised method of machine learning. That is, KCs were previously annotated by human experts, and the machine learning model's performance is measured against the annotations. PRE also has human-annotated relationships from the hypernym/hyponym relationships in WordNet (n.d.). These relationships describe a hierarchical relationship similar to that of PRs. A hypernym represents a superordinate concept whereas a hyponym represents a subordinate concept. WordNet also identifies sister relationships where concepts are related through a common hypernym.

For both KCE and PRE, the following standard machine learning metrics were used: accuracy, recall, precision, and F1 score. Accuracy represents the model's ability to classify correctly an n-gram or concept pair as a KC or PR respectively. However, accuracy is not a useful indicator when a dataset is imbalanced. Out of all the words in a 1,000+ page textbook, very few words are considered KCs. Predicting that each n-gram is not a KC can lead to an accuracy in the high 90s. Precision and recall provide nuance to that score. Recall is a metric that shows how many of the predicted labels were actually predicted. For example, if there were 100 KCs and the model predicted 95 of those, recall would be 95%. Precision is a metric that shows how many of the predictions actually belonged to that class. Using the same example with 100 KCs, if the model predicted all 100 KCs, but also labeled an additional 100 n-grams as KCs, recall would be 95%, but precision is at 50%. Precision and recall make it difficult to compare models, so the F1 score is used to provide a harmonic mean between precision and recall as a singular metric for model comparison. One must use caution when relying on the F-1 metric. It equally weighs both precision and recall when some contexts may prefer one metric over the other. The final evaluation piece is the Zero Rule algorithm which assigns the most frequently occurring classification to all predictions. This serves as a baseline for model performance and is labeled as a dummy classifier.

Results

The KCE and PRE tools were used on several domains. KCE and PRE were used for textbooks on Excel, statistics, history, biology, accounting, chemistry, and calculus. Table 1 shows the domain, the number of pages, chapters, key terms, and concept pairs for each textbook as well as the number of relationships extracted from WordNet. This information can inform the reader of the overall influence each book has on the models. For example, biology had 2,348 key terms—a significant amount more than any other textbook. Due to this influence, the models are compared within and across domains to test the robustness of each model. As a note, two textbooks were used for calculus which is why two numbers are listed in Table 1. When used for the two tasks, the two calculus books were combined and are listed as one domain.

Several models were used and compared for their performance. As both tasks are classification problems, the following models were used and tuned with the listed hyperparameters in Table 2. If a hyperparameter is not listed, the default value from the Python library *sklearn* should be assumed. The defined hyperparameters were found using a grid search method. This method takes a combination of parameters (called hyperparameters) and tests the combinations of each one to determine the settings that create the best model. With regards to the imbalanced datasets mentioned in the Methods section, the *class_weight* hyperparameter for certain models was set to *'balanced'* as a means of oversampling the smaller class.

The remainder of this section will address the primary research questions presented in the Methods section. RQ1 addresses design implications for creating intelligent tutoring services. RQ2 addresses the validation of each task and the models associated with it. Finally, RQ3 highlights the practical benefits of these services.

Research Question 1: Design Implications

Creating a service meant to exist in an ecosystem without the ecosystem is a difficult challenge. This challenge presented several insights. First, what was intended to be one service turned into two separate ones due to the need for modularity and evaluation of the first service (KCE). Second, communication between these services and yet future services becomes problematic. And, third, a service should obviate the need for technical expertise in order to be adopted by a wide audience.

Modularity. The two tasks presented in this paper were intended to be one service. This service, within a larger, modular framework would have been one of several services working together to create an intelligent tutor. As a system is broken down into modular pieces, the question arises: how granular does a service need to be? If an ecosystem already existed, it may be an easier question to answer. As it stands, the decision to create two services stemmed from the nature of the two tasks. KCE and PRE are two closely related tasks; KCs are needed in order to determine PRs. However, the OpenStax books already had the key terms identified. While this proved valuable for creating a supervised model that can be used on books without a glossary, the KCE would not be needed in this case. Instead, another service like a simple web crawler could be used to extract the key terms. This distinction is important to make as it emphasizes the need for a modular framework that fits the context. Having various tools available to fit the textbook in this case would be useful for an author (i.e., being able to use a web crawler and PRE together). If a service only provided KCE and PRE without the option of splitting up the task, the service is no longer usable for that author. The takeaway is to consider the context under which a service is to be used. While a parsimonious design may seem ideal (like it did in this case), the context may dictate something more nuanced. Services can always be combined together, so it might be best to err on the side of greater granularity.

Communication. The second insight stemmed from the first. If the two tasks remained under one service, communication between the two would not be problematic. Services are free to use whatever internal communication methods that fit the situation (Colby & Rich, 2020). When services communicate with each other, a standard needs to be in place to ensure proper transfer of information. Fortunately, many standards already exist. Choosing one is the challenge—an understanding of the needs for each layer is required. Unfortunately, the entire ecosystem does not exist. This leads to a chicken or the egg scenario—does the standard or the service come first? This question needs to be explored further as additional services are created.

Table 1

Descriptive Statistics for Each Domain Used for KCE and PRE

Domain	*# of Pages*	*# of Chapters*	*# of KCs*	*# of Concept Pairs*	*# of Supervised PRs*	*# of Supervised Co-relations*
Excel	252	14	164	539	19	0
Statistics	913	13	150	349	1	3
History	1046	32	409	250	0	1
Biology	1578	47	2348	4016	94	77
Accounting	1055	16	442	955	8	8
Chemistry	1331	21	763	1892	16	17
Calculus 1 & 2	873/829	6/7	195/163	464/445	1/1	2/0
Physics	1418	34	938	2406	29	27
Precalculus	1156	12	316	1020	5	8

Table 2

Machine Learning Models and Hyperparameters for KCE and PRE

Model	***KCE***	***PRE***
K-Neighbors Classifier	n_neighbors=3	algorithm="brute", n_neighbors=10, weights="distance"
SVC	gamma="scale", class_weight="balanced"	c=100, class_weight="balanced"
Random Forest Classifier	N/A	n_estimators=500, max_depth=7, class_weight="balanced"
Extra Trees Classifier	N/A	n_estimators=500, class_weight="balanced"
Gradient Boosting Classifier	loss="exponential", n_estimators=100, max_depth=3	max_depth=7, n_estimators=300

Jargon and adoption. The last insight touches on the issue of creating a technical service for a potentially non-technical audience. In this context, the jargon of machine learning may be difficult for a layperson to understand. If an author is constructing the content layer for their tutor and is presented with several options that fit their context, they would be pleased. When it came down to selecting one service over the other, they may find themselves overwhelmed by the differences between supervised vs. unsupervised models, ranking KCs versus classifying KCs, or the precision versus recall metrics. This problem is not unique to the content layer—choosing between different student modelling techniques presents the jargon issue as well. The benefit of a services-based ecosystem is that the barrier of entry for creating intelligent tutors is reduced; an author does not need expertise in all aspects of intelligent tutoring in order to make practical use of each service. In short, services should be designed in a way that technical expertise and jargon do not interfere with the overall goal of a services-based ecosystem: to make authoring intelligent tutors available to non-experts.

In summary, the three insights can be stated as such: context dictates if tasks can be combined or if they should be separate services, the communication problem between services presents a chicken or egg scenario, and technical expertise and jargon should not get in the way of adoption by non-experts.

Research Question 2: Validating the Services

As was described in the Methods section, human annotation can serve as an evaluation and validation of ontology learning. Thus, human annotated KCs and PRs were used to test the models. KCs were annotated via a glossary for each textbook that the authors identified. PRs were identified via WordNet (n.d.), a hierarchical collection of concepts. Tables 3 and 4 show the performance of each model within and across domains for the task KCE respectively. Table 5 shows similar information as it pertains to the task PRE. Tables 3 through 5 report the following metrics: accuracy (A), precision (P), recall (R), and the F1-score (F1). Due to the small number of relationships extracted from WordNet for each domain, PRE is only measured across domains to ensure enough data points for the models.

Key concept extraction. Three machine learning models were tested independently of each other and then included in a voting classifier. The voting classifier uses the predictions of the three models and chooses the majority vote as the classification. The metrics for these models tell an interesting and consistent story. The SVC model had the highest recall within and between every domain, but the lowest precision for almost all domains. The Gradient Boosting Classifier has the highest precision in all cases within domains, and it has the highest for the majority of cases between domains. The voting classifier maintains the best F1-score for the majority of domains within and between each. The practical benefits of this knowledge is that SVC will tag nearly all the supervised KCs as KCs at the cost of over-identifying KCs, causing the author to sift through potentially extraneous KCs. The Gradient Boosting Classifier may not tag all the supervised KCs, but it does have fewer extraneous KCs to sift through. The voting classifier seems to strike a balance for the majority of use cases. Deciding on which classifier to use is up to the author, but the author may need to be guided through the process of choosing due to jargon. Fortunately, a modular mindset can allow a service to have these preferences in mind. For example, an author can be asked a series of questions like, “Would you like to generate a list of key concepts that is liberal and will choose extra key concepts or one that is more conservative in labeling key concepts at the expense of missing out on some?”

Table 3

Model Results within Domains for KCE

Domain	***SVC***				***K-Neighbors Classifier***				***Gradient Boosting Classifier***				***Voting Classifier***				***Dummy Classifier***			
	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1
Excel	.75	**.54**	**.79**	**.50**	.97	.49	.50	.49	.97	.49	.50	.49	.97	.49	.50	.49	.97	.49	.50	**.50**
Statistics	.96	.63	**.98**	.69	.98	.75	.69	.71	.99	**.83**	.60	.65	.99	.82	.68	**.73**	.99	.49	.50	.50
History	.99	.73	**.99**	.81	.99	.85	.83	.84	.99	**.88**	.85	.86	.99	.85	.90	**.87**	.99	.50	.50	.50
Biology	.97	.80	**.97**	.86	.98	.90	.91	.91	.99	**.96**	.93	**.94**	.99	.91	.95	.93	.95	.47	.50	.49
Accounting	.95	.64	**.97**	.71	.98	.78	.73	.75	.98	**.84**	.65	.71	.98	.78	.78	**.78**	.98	.49	.50	.49
Chemistry	.95	.70	**.97**	.77	.98	.80	.79	.80	.98	**.84**	.81	.82	.98	.79	.87	**.83**	.97	.48	.50	.49
Calculus	.92	.60	**.96**	.65	.98	.78	.70	.73	.99	**.91**	.69	**.76**	.98	.81	.73	**.76**	.98	.49	.50	.50
Physics	.96	.71	**.98**	.76	.98	.81	.79	.80	.98	**.87**	.86	**.86**	.98	.82	.90	**.86**	.97	.49	.50	.49
Precalculus	.84	.58	**.91**	.59	.97	.79	.58	.62	.97	**.82**	.62	.67	.97	.81	.66	**.71**	.97	.49	.50	.49

Regarding the domains themselves, another interesting note is the performance on the math-focused domains: statistics, accounting, calculus, and precalculus. Within and between domains, these have the lowest F1-scores for the voting classifier. When describing the knowledge representation sublayer, we described how the knowledge itself dictates how it needs to be represented (Colby & Rich, 2020). Math domains, typically more procedural in nature, may be better suited with production rules (Anderson, Boyle, Corbett, & Lewis, 1990) as a form of knowledge representation. These rules capture the how-to of the domain. Certainly, concepts exist in math domains, but the nature of the knowledge and how it is presented could be

Table 4

Model Results between Domains for KCE

Domain	***SVC***				***K-Neighbors Classifier***				***Gradient Boosting Classifier***				***Voting Classifier***				***Dummy Classifier***			
	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1
Excel	.77	**.53**	**.76**	.50	.97	.51	.50	.50	.97	**.53**	.51	**.51**	.97	.52	.51	**.51**	.98	.49	.5	.49
Statistics	.96	.63	**.98**	.70	.98	**.68**	.71	**.69**	.98	.60	.54	.56	.98	.67	.72	**.69**	.99	.49	.50	.50
History	.99	.70	**.99**	.78	.99	.87	.68	.74	.99	**.97**	.67	.75	.99	.89	.74	**.80**	.99	.50	.50	.50
Biology	.97	.81	**.98**	**.88**	.97	.90	.72	.78	.97	**.94**	.67	.74	.97	.91	.77	.82	.95	.47	.50	.49
Accounting	.95	.64	**.97**	.70	.97	**.70**	.86	.76	.97	**.70**	.91	**.77**	.96	.68	.94	.75	.95	.47	.50	.49
Chemistry	.95	.70	**.97**	.78	.97	.77	.80	.78	.98	**.84**	.88	**.86**	.97	.79	.90	.83	.97	.48	.50	.49
Calculus	.92	.61	**.96**	.66	.96	**.66**	.81	**.70**	.96	.65	.84	**.70**	.96	.65	.84	**.70**	.98	.49	.50	.49
Physics	.96	.72	**.98**	.79	.98	.83	.80	.81	.98	**.87**	.84	**.86**	.98	.84	.88	**.86**	.97	.49	.50	.49
Precalculus	.86	.57	**.92**	**.59**	.96	.59	.57	.58	.97	**.64**	.57	**.59**	.96	.59	.59	**.59**	.98	.49	.50	.49
All Domains	.95	.67	**.96**	.74	.98	.82	.79	.80	.98	**.85**	.82	**.84**	.98	.81	.87	**.84**	.97	.49	.50	.49

Note: Domain listed was left out. All domains did not leave any domains out.

encapsulated more accurately in procedural knowledge. This could be the same with the problem-solving heuristics of a domain which cannot be captured simply as KCs.

The Excel domain, which is very much procedural and technical in nature, follows the same pattern of poorer performance when predicting key terms. In fact, the domain hardly performed better within and between domains than the dummy classifier. This is probably exacerbated by the fact that the Excel textbook does not follow the same format as the OpenStax textbooks. This suggests that an approach based on the structure of a text may yield varying results based on how structured the text is. Additionally, the idea that procedural domains benefit more from a rules-based knowledge representation is supported by the underlying knowledge structure of the Excel class. Assignments are graded by a rule engine that assesses whether a student demonstrates adequate knowledge of a task. A simpler rule engine was implemented as a feedback mechanism which allowed students to receive immediate feedback. Thus, rules seem to be more appropriate for measuring performance on a task rather than testing knowledge of a concept.

Prerequisite relationship extraction. Table 5 tells a less confident story with regards to the PRE task. SVC continues to have the highest recall between domains. The K-Neighbors Classifier more often has the highest precision. The best F1-scores are spread across K-Neighbors, SVC, and Extra Trees Classifier depending on the subject. The Gradient Boosting Classifier, overall, performed worse than the other models. However, the performance of the task when predicting between domains is not great. The greatest F1-score spread between a model and the dummy classifier was only 16 points. It does not seem as if the models do well at predicting new material. Interestingly, when the models are trained on all domains, the metrics are at a more acceptable and usable range.

Part of this poor performance may come from WordNet, the supervision source. In total, only 154 concept pairs were identified as having a PR and 143 with corequisite relationships out of 11,797 total concept pairs. WordNet may not accurately capture PRs and thus may be an inadequate tool to use for this task. A possibility is that it could be better defined for some domains like biology (which had the most PRs identified of any subject) than others. A similar concern was expressed by Liang et al. (2015) with regards to using Wikipedia for PRE. It is also possible that the number of strict PRs that would exist in an ontology are quite few. For example, five experts annotated PRs for a computer science textbook with 353 concepts (Alzetta et al., 2019). Of the concept pairs, only 25 pairs were identified as having a PR by all five experts, 46 were agreed upon by four experts, 83 by three, 214 by two, and 698 by just one annotator. Alzetta et al. (2019) included all annotations regardless of how many experts agreed. In this case, it seems that annotating PRs is less about an agreed upon standard, and more about the expert's own experience. Another example relied on only one annotator, citing the need for consistency in annotation for the machine learning models (Changuel, Labroche, & Bouchon-Meunier, 2015).

Perhaps this suggests that PRE is less of a science and more of an art. However, other work has achieved better results (with a larger set of human-annotated PRs) (De Medio et al., 2016; Gasparetti et al., 2018; Liang et al., 2015; Miaschi et al., 2019). Several of these works relied on Wikipedia as an outside structure to determine if PRs exist. Additional features outside a textbook are considered (e.g., how concepts link to each other or the inherent categorization of articles on Wikipedia). While the results in these studies are better, Wikipedia is cumbersome to work with.

While processing speed for these tasks is discussed in the next section, it is appropriate to note here that adding a Wikipedia check for links, page info, etc. slows the process significantly if relying on API calls. An alternative is to download all of Wikipedia and querying it locally, but that requires 10s of gigabytes of space. The purpose of this research was to rely on the textbook as much as possible with WordNet as a tool for supervision. However, due to the seemingly subjective nature of PRs, relying on crowdsourced knowledge bases like Wikipedia may be an appropriate method to detect PRs. What is needed, though, is a cumulative knowledge of those PRs. Once those PRs are identified, the computing effort has been done. Storing the relationships in an accessible repository (e.g., perhaps a library like WordNet) would allow another service to query it and get the needed information. Such an approach may be better suited for a service than an automated attempt at extracting the relationships.

One drawback of relying on such external sources, though, is evident in the performance on the Excel domain. Seventeen of the 19 PRs would not be found on Wikipedia as they involve concepts that represent Excel formulas. These formulas would not make it past a Wikipedia filter and would be left out. Thus, key concepts and their relationships could be excluded if they represent jargon of the domain that has not made it into a Wikipedia article.

Research Question 3: Utility

Determining the utility of a service is subjective in nature based on the input of end users. Without users to test the services in this paper and to provide feedback, it is difficult to ascertain if these services are useful. However, some questions can be asked that address the utility of these services. First, how lengthy of a process is it to use these services? Second, what does the process look like from the user's perspective? And, third, how does it fit in the larger ecosystem?

Creating the domain has always been a time-intensive task for ITSs. A goal of a services-based ecosystem is to reduce the development costs of an ITS overall. For that reason, a big question for content layer services is if they can reduce the time it takes to create the domain. This paper deals with the structure of the domain rather than the underlying content. The structure is what is needed to coordinate with other services. No clear indications have been identified in the literature that give an idea of how long the tasks presented in this paper should take. Without such a benchmark, it is difficult to gauge if the automated processes are considered fast or slow. Regardless, estimated times for each task are presented in Table 6.

Table 5

Model results between domains for PRE

Domain	*K-Neighbors Classifier*				*SVC*				*Random Forest Classifier*				*Extra Trees Classifier*				*Gradient Boosting Classifier*				*Voting Classifier*				*Dummy Classifier*			
	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1	A	P	R	F1
Excel	.97	**.33**	.32	**.33**	.43	.33	.14	.20	.97	**.33**	.32	**.33**	.97	**.33**	.32	.33	.97	**.33**	.32	**.33**	.97	**.33**	.32	**.33**	.90	.30	**.33**	.32
Statistics	.91	.39	.46	.41	.41	.37	**.65**	.26	.94	**.46**	.47	**.46**	.93	.41	.46	.42	.94	.42	.49	.45	.93	.41	.46	.43	.89	.30	.33	.31
History	.79	**.89**	.56	.55	.35	.63	**.58**	.34	.74	.60	.57	.57	.74	.41	.38	.39	.76	.38	.50	.43	.74	.60	.57	**.57**	.76	.38	.50	.43
Biology	.83	**.56**	.40	.42	.58	.48	**.60**	**.46**	.81	.48	.40	.42	.80	.47	.41	.42	.83	.52	.37	.38	.81	.50	.42	.44	.83	.28	.33	.30
Accounting	.93	**.47**	.39	**.42**	.60	.41	**.63**	.38	.94	.43	.36	.37	.93	.46	.37	.38	.94	.40	.36	.36	.94	.44	.37	.39	.96	.32	.33	.33
Chemistry	.86	**.47**	.45	**.45**	.55	.45	**.72**	.43	.85	.42	.39	.40	.85	**.47**	.44	**.45**	.85	.41	.40	.40	.85	.45	.43	.44	.89	.30	.33	.31
Calculus	.93	.38	.41	.39	.72	.42	**.86**	**.43**	.95	.32	.33	.32	.95	**.45**	.41	**.43**	.94	.32	.33	.32	.94	.43	.41	.42	.95	.32	.33	.33
Physics	.87	.54	.43	**.46**	.65	.46	**.59**	**.46**	.87	**.56**	.40	.43	.86	.54	.41	.44	.87	.52	.40	.42	.86	.54	.41	.44	.87	.29	.33	.31
Precalculus	.84	**.49**	.38	**.40**	.54	.40	**.66**	.35	.89	.40	.34	.34	.86	.39	.35	.34	.89	.30	.32	.31	.86	.42	.35	.34	.92	.31	.33	.32
All Domains	.93	.85	.68	.74	.56	.49	**.76**	.46	.93	.80	.68	.73	.93	.79	.71	**.75**	.93	**.88**	.64	.71	.93	.79	.70	.74	.89	.30	.33	.31

Note: Domain listed was left out. All Domains included every domain.

Besides time, another measure of utility is how much work is created for the user. The pipeline for the two tasks in this paper, as shown in Figures 2 and 3, is to extract KCs, allow space for the author to refine the list if needed, then extract the PRs, again allow for refinement before finally translating the resultant concept map into a machine-readable format. It is in those refinement stages where the balance between precision and recall needs to be found, as discussed above. To add to that discussion, it seems that a default perspective to take is to err on the side of higher recall over higher precision if there needs to be a trade-off. The reasoning being that it's

Table 6

Approximate Processing Time to Extract Chapter and Feature Information by Domain (Entire Textbook) for Each Task

Domain	***Chapter Processing***	***KCE***	***PRE***	***Total***
Excel	00:00:02	00:10:43	00:02:22	00:13:07
Statistics	00:00:03	00:06:29	00:06:20	00:12:52
History	00:00:09	00:23:54	00:00:10	00:24:13
Biology	00:00:14	00:29:54	00:20:05	00:50:13
Accounting	00:00:06	00:09:30	00:30:35	00:40:11
Chemistry	00:00:09	00:16:11	00:25:45	00:42:05
Calculus	00:00:09	00:11:41	00:11:03	00:22:53
Physics	00:00:15	00:18:17	01:06:54	01:25:26
Precalculus	00:00:10	00:07:16	01:25:04	01:32:30

better to cover all KCs and sort through some extraneous results than it is to miss some potentially crucial KCs. It is also possible that the lower precision is the result of possible KCs existing in the text that were overlooked when the glossary was made for the supervised datasets.

The last point for this RQ touches on the ecosystem that does not exist yet. When evaluating a service on its utility, how it fits in the overall ecosystem is an important consideration. For example, how would the introduction of this service affect the workflow of the content layer? If the generation of an ontology was done in one service, the introduction of these two services would break that up. Each layer/sublayer will need its own pipeline, and services will need to fit that pipeline unless the pipeline is altered.

Conclusion

This article presented a proof of concept for a service belonging to a modular architecture. The services created in this article were part of the knowledge representation sublayer of the content layer (Colby & Rich, 2020). Two services were created for two tasks: one to handle key concept extraction (KCE) and another for prerequisite relationship extraction (PRE). The KCE task saw promising results for extracting key concepts (KCs) for both within domains and between domains. However, some subjects seemed better suited for the task. Math-related domains had the poorest performance which may be an indicator of needing to represent knowledge in different ways (e.g., production rules) for domains that are more procedural in nature. The PRE task did not perform as well. This leads to a couple conclusions. First, the features extracted from the textbooks may be ill-suited and may require an external source to add some structure to the relationships between concepts. Second, the nature of prerequisite relationships (PRs) may be too subjective for an automated tagging process, needing to rely on human annotators (and a database of human-made decisions) for PR identification.

From creating these services, a few design implications were gleaned. First, the layers framework promotes modularity. When creating a service, an author needs to determine if there is a need for modularity within the service. Second, communication between services is vital and relies on a standard; however, communication within a service can fit the needs of the context. Third, services are meant to be used by a non-expert audience. Descriptions of and use of the service should be as minimal a barrier of entry as possible.

Lastly, a service should make the process of creating an ITS easier and faster. An estimate of how long each task takes was provided in this paper as a comparison marker. Second, services exist within a pipeline for each layer. While the full ecosystem does not exist yet, a service should not disrupt the pipeline in such a way that it becomes more difficult.

In conclusion, this proof of concept had some successes and some areas that need improvement. However, it marks the first step in creating a services-based ecosystem under which ITS can be created and, hopefully, widely adopted.

References

Adorni, G., Alzetta, C., Koceva, F., Passalacqua, S., & Torre, I. (2019). Towards the identification of propaedeutic relations in textbooks. *Lecture Notes in Computer Science Artificial Intelligence in Education*, 1–13. doi: 10.1007/978-3-030-23204-7_1

Alzetta, C., Miaschi, A., Adorni, G., Dell'Orletta, F., Koceva, F., Passalacqua, S., & Torre, I. (2019). Prerequisite or not prerequisite? That's the problem! An NLP-based approach for concept prerequisites learning. *Proceedings of the Sixth Italian Conference on Computational Linguistics (CliC-it 2019).* Retrieved from http://pages.di.unipi.it/miaschi/publication/conference-paper/clic_2019/

Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, *42*(1), 7-49.

Asim, M. N., Wasim, M., Khan, M. U. G., Mahmood, W., & Abbasi, H. M. (2018). A survey of ontology learning techniques and applications. *Database*, *2018*, 1-24. doi: 10.1093/database/bay101

Brooks, C., Winter, M., Greer, J., & McCalla, G. (2004). The Massive User Modelling System (MUMS). *Intelligent Tutoring Systems Lecture Notes in Computer Science,* 635-645. doi:10.1007/978-3-540-30139-4_60

Brusilovsky, P., Sosnovsky, S., & Shcherbinina, O. (2005). User modeling in a distributed e-learning architecture. *User Modeling 2005 Lecture Notes in Computer Science,* 387-391. doi:10.1007/11527886_50

Changuel, S., Labroche, N., & Bouchon-Meunier, B. (2015). Resources sequencing using automatic prerequisite—outcome annotation. *ACM Transactions on Intelligent Systems and Technology*, *6*(1), 1–30. doi: 10.1145/2505349

Colby, B. R., & Rich, P. (2020). *From systems to services: Changing the way we conceptualize ITSs* (Unpublished doctoral dissertation). Brigham Young University, UT.

De Medio, C. D., Gasparetti, F., Limongelli, C., Sciarrone, F., & Temperini, M. (2016). Automatic extraction of prerequisites among learning objects using Wikipedia-based content analysis. *Intelligent Tutoring Systems Lecture Notes in Computer Science*, 375–381. doi: 10.1007/978-3-319-39583-8_44

Gasparetti, F., De Medio, C., Limongelli, C., Sciarrone, F., & Temperini, M. (2018). Prerequisites between learning objects: Automatic extraction based on a machine learning approach. *Telematics and Informatics, 35*(3), 595-610.

Gibbons, A. S. (2014). *An architectural approach to instructional design*. New York, NY: Routledge.

Hasan, K. S., & Ng, V. (2014). Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1262-1273). Baltimore, MD: Association for Computational Linguistics. doi: 10.3115/v1/p14-1119

Heffernan, N. T., Turner, T. E., Lourenco, A. L., Macasek, M. A., Nuzzo-Jones, G., & Koedinger, K. R. (2006). The ASSISTment builder: Towards an analysis of cost effectiveness of ITS creation. In G. Sutcliffe, & R. Goebel (Eds.), *FLAIRS Conference* (pp. 515-520). Melbourne Beach, FL: Association for the Advancement of Artificial Intelligent (AAAI) Press.

Ismail, K. N., Ahmadon, F., & Shahbudin, F. E. (2018). Instructional material development using ontology learning. *Social and Management Research Journal, 15*(2), 35-46. doi:10.24191/smrj.v15i2.4969

Jarvis, M. P. (2004). *Applying machine learning techniques to rule generation in intelligent tutoring systems* (Unpublished master's thesis). Worcester Polytechnic Institute, MA.

Kleinberg, J. (2002). Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery, 7*, 373-397 doi: 10.1145/775047.775061

Liang, C., Wu, Z., Huang, W., & Giles, C. L. (2015). Measuring prerequisite relations among concepts. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1668-1674). Lisbon, Portugal: Association for Computational Linguistics. doi: 10.18653/v1/d15-1193

Miaschi, A., Alzetta, C., Cardillo, F. A., & Dell'Orletta, F. (2019). Linguistically-driven strategy for concept prerequisites learning on Italian. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications* (pp. 285-295). Florence, Italy: Association for Computational Linguistics. doi: 10.18653/v1/w19-4430

MyEducator (n.d.). Retrieved from https://www.myeducator.com/

Nye, B. D. (2015). AIED is splitting up (into services) and the next generation will be all right. *AIED Workshops, 1432*(4), 62-71.

OpenStax. (n.d.). Retrieved from https://openstax.com/

Parameswaran, A., Garcia-Molina, H., & Rajaraman, A. (2010). Towards the web of concepts. *Proceedings of the VLDB Endowment*, *3*(1-2), 566–577. doi: 10.14778/1920841.1920914

spaCy. (n.d.). Retrieved from https://spacy.io/

Suraweera, P., Mitrovic, A., & Martin, B. (2005). A knowledge acquisition system for constraint-based intelligent tutoring systems. *12th International Conference on Artificial Intelligence in Education (AIED 2005)* (pp. 638-645). Amsterdam, the Netherlands: IOS Press. http://hdl.handle.net/10092/348

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist, 46*(4), 197-221. doi:10.1080/00461520.2011.611369

Wang, S., & Liu, L. (2016). Prerequisite concept maps extraction for automatic assessment. In *Proceedings of the 25th International Conference Companion on World Wide Web - WWW 16 Companion (pp. 519-521)*. Montreal, Canada: International World Wide Web Conferences Steering Committee. doi: 10.1145/2872518.2890463

WordNet. (n.d.). Retrieved from https://wordnet.princeton.edu/

DISSERTATION CONCLUSION

The first article presented a new framework for a services-based ecosystem of intelligent tutoring. This new framework can help shift the thinking of ITS authors from creating *systems* to creating *services*, while embracing the modularity inherent in a services-based economy. The current architectures used to develop ITSs are too rigid in their approach, as has been demonstrated by decades of research resulting in paradigmatic silos at the cost of a cumulative history. Based on Gibbons' design layers (2014), the article provides a framework that guides the operationalization of the many services needed to design, develop, maintain, and ultimately proliferate effective ITSs. It also provided the basis for the second article, which was a proof-of-concept for services belonging to the content layer.

The services created in the second article were part of the knowledge representation sublayer of the content layer. Two services were created for two tasks: one to handle key concept extraction (KCE) and another for prerequisite relationship extraction (PRE). The KCE task saw promising results for extracting key concepts (KCs) for both within domains and between domains. However, some subjects seemed better suited for the task, specifically those that are more declarative in nature than procedural. The PRE task did not perform as well. This leads to a couple conclusions. First, the features extracted from the textbooks may be ill-suited and may require an external source to add some structure to the relationships between concepts. Second, the nature of prerequisite relationships (PRs) may be too subjective for an automated tagging process, needing to rely on human annotators (and a database of human-made decisions) for PR identification. Additional insights were gleaned based on the design process of creating these services, the primary of which being: services should be designed in such a way that a non-expert author can make use of it to create an intelligent tutor.

DISSERTATION REFERENCES

Colby, B. R. (2017). *A comprehensive literature review of intelligent tutoring systems from 1995-2015*. (Publication No. 7239) [Master's Thesis, Brigham Young University]. ScholarsArchive.

Gibbons, A. S. (2014). *An architectural approach to instructional design*. New York, NY: Routledge.

Mendicino, M., Razzaq, L., & Heffernan, N. T. (2009). A comparison of traditional homework to computer-supported homework. *Journal of Research on Technology in Education, 41*(3), 331-359.

Nye, B. D. (2015). AIED is splitting up (into services) and the next generation will be all right. *AIED Workshops, 1432*(4), 62-71.