FLUENT
Fast Learning from Unlabeled Episodes of Next-Generation Tailoring

TLA Specification Requirements
Contract: W911QY-16-C-0019

Prepared for:
Dr. Sae Schatz, Director
Advanced Distributed Learning (ADL) Initiative

Prepared by:
SoarTech

3600GreenCt.,Ste.600,Ann Arbor, MI48105
Phone: 734-327-8000 • Fax: 734-913-8537 • Web: www.soartech.com
**TABLE OF CONTENTS**

Table of Contents ........................................................................................................................................... 2

Revision History ................................................................................................................................................ 3

1 Introduction .................................................................................................................................................. 3

2 TLA Specification Requirements Summary ............................................................................................ 3

3 Design-Based Research Requirements Analysis ....................................................................................... 5

3.1 Use Case Inclusive and Forward Thinking ............................................................................................. 5

3.1.1 Many Users ......................................................................................................................................... 6

3.1.2 Many Deployment Platforms ............................................................................................................. 6

3.1.3 Many Protocols .................................................................................................................................. 6

3.1.4 Many Components ............................................................................................................................ 7

3.2 Longevity Techniques ............................................................................................................................. 7

3.3 Encourage Community Collaboration .................................................................................................... 8

3.3.1 Build On Existing Work .................................................................................................................... 8

3.3.2 Easy to Use ....................................................................................................................................... 9

3.4 Data Centric ............................................................................................................................................. 11

3.4.1 Lots of Data ....................................................................................................................................... 11

3.4.2 Plug and Play .................................................................................................................................... 12

3.4.3 Producers and Consumers ............................................................................................................... 12

3.4.4 Extensible ......................................................................................................................................... 13

3.4.5 Real-time Data Processing and Stored Data History ......................................................................... 14

3.4.6 Distributed Data Models .................................................................................................................. 14

3.4.7 Performance Minded ......................................................................................................................... 15

3.4.8 Automation Friendly ......................................................................................................................... 15

3.4.9 Security ............................................................................................................................................ 16
1 INTRODUCTION
The TLA is an architecture to enable learning that is described by specifications. See the TLA Design Document.docx for an introduction to the TLA architecture. Many terms and definitions are provided in the TLA Design Document.docx as well, so the authors suggest that readers reference that document as needed. TLA specifications are formalized documentation that enables multiple providers to contribute individual components to a TLA ecosystem in such a way that all components interoperate seamlessly. Specifications are aimed at long-term goals, and, thus, typically include enough flexibility to support many instantiations that support different use cases. TLA development is using a design-based research approach; iterative analysis, design, development, and implementation cycles will shepherd the initial seed of an idea through to maturation. This document describes the requirements that the TLA specifications need to address, as elicited by the design-based research approach. The initial set of requirements is an outcome of analysis of the Year 1 experimental prototype demonstrated at Fort Bragg in April 2017.

2 TLA SPECIFICATION REQUIREMENTS SUMMARY
This section briefly summarizes the high-level TLA specification requirements identified during Year 1 of the TLA design-based research. The summary in this section is presented up front for readers desiring a quick overview. For a deeper understanding, please continue on to section 3. For ease of reference, each summary table in this section corresponds to a major topic in section 3. Please note that the order in which the requirements are summarized in this section reflects the order of discussion in the analysis section; ordering here does not imply relative importance.

<table>
<thead>
<tr>
<th>Summary of Related Requirements: Use Cases and Forward Thinking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoupled from specific use cases</td>
</tr>
<tr>
<td>Support many types of users without modification each time a new user type is identified</td>
</tr>
<tr>
<td>Support many types of deployment platforms without modification each time a new platform type is introduced</td>
</tr>
<tr>
<td>Support many types of transport protocols</td>
</tr>
<tr>
<td>Support many types of security protocols</td>
</tr>
<tr>
<td>Support inclusion of more than one transport protocol in a single TLA instantiation</td>
</tr>
<tr>
<td>Support inclusion of more than one security protocol in a single TLA instantiation</td>
</tr>
<tr>
<td>Decouple data definition from transport and security protocol description</td>
</tr>
<tr>
<td>Decoupled from specific, instantiated, component types</td>
</tr>
<tr>
<td>Support adding new component types without specification modification</td>
</tr>
<tr>
<td>Easy to maintain</td>
</tr>
<tr>
<td>Subdivided into families of related specifications to facilitate modularity and long term maintenance</td>
</tr>
<tr>
<td>Easy to extend and refine</td>
</tr>
<tr>
<td>Compatible with a &quot;proving ground&quot; workflow that facilitates improvement</td>
</tr>
<tr>
<td>Compatible with best practices that bridge the gap between long lived specifications and short-lived implementation details</td>
</tr>
</tbody>
</table>

**Summary of Related Requirements: Encourage Community Collaboration**

| Leverage existing standards and specifications whenever possible to meet requirements |
| Support combining existing specifications (including parts of existing specifications) |
| Support recording provenance for included, pre-existing, specifications |
| Support compatibility with many existing specifications |
| Support compatibility with overlapping existing specification through equivalency mapping or other similar means |
| Encourage community contribution in the workflow for extensions and improvements |
| Easy to understand |
| Easy for developers to use |
| Human readable |
| Machine interpretable |
| Support automated component discovery |
| Support computational resources residing in the cloud, or on a traditional organizationally owned server |
| Support negotiation of resource limits when new components are "plugged in" |
| Support cost based resource throttling |
| Support communication of component QoS capabilities and needs |
| Support fault tolerance |
| Support respecting learner privacy |
| Support data removal |

**Summary of Related Requirements: Data Centric**

| Support a data centric approach |
| Use a small number of easily understood logical data models |
| Use a non-component centric data organization paradigm so that adding use cases does not result in an explosion of logical data models |
| Support logical data models that are capable of sharing large quantities of data across many functional areas |
| Support sharing data about science of learning concepts and software assets in addition to data about the learner experiences |
| Support utilization of many science of learning underlying theories |
| Support utilization of many content sequencing strategies |
Support a producer/consumer data flow model
Clearly describe data concept: What data is shared?
Clearly describe data structure: How is the shared data structured?
Clearly describe conceptual API: What can be said about the data between producers and consumers?
Decouple data concept, data model and conceptual API from specific use case specific assumptions
Support adding new data types to support new use cases without requiring a specification modification each time a new piece of data is identified
Support Conceptual API flexible enough to not require a specification modification each time a new use case is introduced
Support developer friendly API for requesting filtered data from data producers
Support storing data history to support non-real-time data analysis
Support both real-time data and non-real-time data exchanges
Support metadata for data producers to advertise the parts of the logical data models they produce
Support metadata for data producers to advertise whether they keep a historical record of the data they produce for consumers to retrieve at a later time
Support a distributed data model, where the data can reside in different physical locations, but be accessed by all data consumers as though it were a single, globally shared data model
Support distributing the logical data model such that data producers that can provider history are not required to transfer the data to a physically centralized data storage location
Support multiple many types of scale out techniques without requiring a specification modification
Support many types of load balancing techniques without requiring a specification modification
Support data filtering that optimizes network bandwidth use
Support programatically adding and modifying metadata in a way makes it easy to automate metadata population
Support many metadata population automation techniques
Decouple data definition from descriptions of securing at rest data

3 DESIGN-BASED RESEARCH REQUIREMENTS ANALYSIS
This section provides more detailed descriptions of the requirements introduced in section 2. It describes how each requirement was identified during Year 1 of TLA design-based research.

3.1 USE CASE INCLUSIVE AND FORWARD THINKING
It isn't enough to produce specifications that meet today's requirements; TLA specifications should be relevant for many years to come. Because the desired lifespan of the TLA is long (aimed at 2025), it is necessary to consider how to stay relevant in the face of emerging instructional methodologies and learner needs. To ensure the TLA lifespan is numbered in years or decades rather than months, one of the core design requirements is that it must be designed such that new and unexpected use cases are encouraged and welcomed. Assuming that the number of use case is not fixed means that the TLA must be designed to be flexible and adaptable. How does this impact specifications describing the TLA? The most significant impact is that assumptions coupled to specific use cases cannot be baked into the specification. Abstractions that are applicable to many use cases must be elicited and utilized so that the TLA specifications do not need to change each time a new use case is introduced. In this section, we will explore some of the dimensions that require abstraction to maintain flexibility across use cases.

TLA Specification Requirements
Version: 0.1.1
3.1.1 Many Users

If there is not a fixed use case, then it follows that there can be many different users. Some of the types of users that Year 1’s research has identified include:

- Learners
- Content Curators
- Researchers
- Instructors
- Network/IA Administrators

What does this mean for the TLA specifications? Since there is no single user-facing application that will interface with the TLA, it is not possible to enumerate all of the ways that the user-facing application(s) will want to use the data that the TLA makes available. Therefore, the specification must be designed such that different user-facing applications with different goals can be constructed without a specification change when a new user-facing application is needed. Examples of such applications include learning activities and instructor dashboards.

3.1.2 Many Deployment Platforms

To meet the needs of a variety of use cases, the TLA specifications should be inclusive in regard to deployment platforms. We live in a world of constantly changing hardware and devices, and the consumer device market changes especially rapidly. To be effective over time, the TLA specifications must be capable of supporting the end user’s desired hardware delivery devices. Today that might mean laptops, tablets, and cell phones. But over the lifetime of the TLA, it must be extensible to include new platforms as they emerge, for delivery devices in smart cars or smart home appliances. To view an animated depiction of how a user might interact with the TLA in the future, follow this link:


When a new delivery device emerges, it should not require a TLA specification modification to begin using it with a TLA instantiation.

3.1.3 Many Protocols

Specifications are designed to help developers communicate. This often requires a great deal of specificity. While specificity facilitates near-term communication, it can impede long-term maintenance. If a use case is tightly defined, then assumptions can be made that simplify how to address the level of specificity that is required. This is particularly prevalent in regard to transport protocols and security protocols. For example, a tightly constrained use case might be able to make the simplifying assumption "all software communication necessary can be accomplished using REST." When this is the case, the specifications for addressing the needs of such use cases are often tightly coupled to a specific transport protocol, such as REST. This conflates the description of what the data is and what can be said about it with the method used to transport it across the wire. Similarly, security protocols, or how to secure the data being transported, can be conflated with what the data is and what can be said about it. This works reasonably well for systems whose use cases are not expected to vary over time. However, since the TLA is designed to welcome new and unexpected use cases, these kinds of simplifying assumptions do not satisfy long-term requirements. As an illustration, let’s consider the choice of REST as the transport protocol for the Year 1 experimental prototype and initial draft of the TLA specifications. REST is
fundamentally a pull protocol; when a component wants information, it asks for it directly. The Fort Bragg experiment revealed that in addition to pull, pushing is also needed to satisfy the meta-adaptation use case. Some of the component vendors would like to register interest in an event happening and receive a notification if that event takes place (push). In particular, the Fort Bragg experiment elicited the need to mix push and pull protocols in the same implementation. To meet the flexibility needs that different use cases demand, the transport and security protocols must be decoupled from the description of what the data is and what can be said about the data in the TLA specifications.

3.1.4 Many Components
Designing an architecture that can be adapted to different use cases means that it must provide building blocks that can be composed into many different instantiations, where each instantiation is targeted at a specific set of use cases. If the specification is component-centric, then it would need to be modified each time a new type of component was discovered to be necessary to meet the needs of a specific use case. Instead specifications should support flexible component descriptions so that new component types that conform to the TLA specifications can be added as necessary in individual instantiations. Based on the Year 1 experimental prototyping and analysis of a variety of potential future use cases, it appears that all types of components can be categorized as

- TLA Conformant Apps – User-facing tools that use TLA data to meet an end user need
- Processors – Software components that intake data, transform it in some way, and output another type of data
- Data Stores – Software components that store, but do not modify data

The TLA specification should, therefore, provide a means for each of these general categories of components to utilize TLA data, without coupling the specification to a specific type of component such as an "Evidence Mapper." That is, the TLA specification cannot be tightly coupled to specific component names.

3.2 Longevity Techniques
To achieve the vision of a long-standing TLA specification that is easy to maintain, several known techniques can be employed. As discussed in the preceding sections, identifying abstractions and decoupling the specification from near-term implementation choices can be very beneficial. However, in practice, the specification still must allow developers to communicate well enough to achieve functional implementations. Fortunately, two specific strategies make this possible.

1. Best Practices guides can be published that fill in practical details that indicate the best-known way to implement a specification using today's technology. As technologies change, if the specification was well-thought-out, it will not need to be modified for each new technology that emerges. Instead, the best practices can be updated to describe how to utilize newly emerged technologies to implement the existing specification.

2. Specifications can be divided into families of related specifications. Implementations may then utilize the subset of specifications that best meets their needs. For example, isolating the description of what the data is in one specification and providing separate specifications that describe how to use individual transport protocols with the data in a referenced data
specification can allow implementers to select the data they need without an implied transport protocol selection.

Lastly, to succeed at long-term viability, the TLA specification must have a means of interacting successfully with the community to enable refinement over time. To do so, a workflow must be developed that supports a "proving ground" for identification of new ideas, evaluation of their merit, and promotion of best-of-breed ideas from the proving ground into the formal specifications.

### 3.3 Encourage Community Collaboration

The goal of the TLA is to create an ecosystem that fosters collaboration. Instead of building training solutions by contracting a single solution provider to build the full product, ADL envisions an environment where many vendors can plug in components that reflect their particular areas of expertise, thereby creating a learning system that surpasses what can be created by a single provider. To bring this vision to fruition, the TLA specifications must be community oriented. They must be publicly available for many collaborators to use and contribute to. In this section, we will explore some of the dimensions required to encourage community collaboration.

#### 3.3.1 Build On Existing Work

The TLA specifications should not start from scratch. Leveraging existing standards and specifications whenever possible to meet requirements is a primary goal. Doing so will allow us to take advantage of years’ worth of work the community has put into developing them. In Year 1’s experimental prototype, existing standards were not incorporated. This was a deliberate, and temporary, choice. The requirements at the start of Year 1 were insufficiently defined to be matched to existing specifications. Allowing free exploration of ideas during Year 1 was done mindfully, with the end goal of eliciting requirements. During the creation of the initial experimental prototype and its demonstration at Fort Bragg in April 2017, many refinements were made to our initial understanding of TLA requirements. The first draft of the TLA API Specifications (version 0.1) was produced as a basic capture of what was necessary at an engineering level to produce a fully functional TLA instantiation. These preliminary APIs have been used by the requirements analysis process as input and have been very helpful in identifying requirements that may otherwise have been overlooked. Input has also been gathered in other ways, such as during the 2016/2017 Delphi study. We have now gathered enough information to allow us to step back, analyze what we learned, and write down the requirements that will need to be formalized into specifications (this document). The next step is to perform a gap analysis where we evaluate the TLA specification requirements we elicited over the course of the first year against the wide variety of existing specifications out in the community. A gap analysis will help us to understand which existing specifications we can leverage, and where the TLA has unique requirements not yet met by any existing specification.

To foster a community collaboration, the TLA specification should be assembled by combining existing standards that match TLA requirements. Doing this effectively will be a challenge, but there are several goals that should guide the end result. While some specifications may be appropriate to include wholesale, there will be others that only partially overlap the TLA’s requirements. In the case of partial overlap, a methodology for incorporating the partial match must be identified (such as Metadata Application Profiles). We do not want to invent a new standard that overlaps part of an existing one. When existing standards are included wholly or partially, provenance must be tracked so the origin of

---

*TLA Specification Requirements*

*Version: 0.1.1*
the inclusion can be traced. Next, there will be places where more than one existing standard meets a TLA requirement. Where there are already existing, overlapping specifications, the goal for the TLA specification should be to build on our spirit of inclusivity. Rather than down-selecting to a single match, every effort should be made to make the TLA compatible with all matches. This may entail producing an equivalency mapping that facilitates converting terminology (and underlying principles) from one specification language to another. Lastly, as the TLA is forging new ground in learning systems, there will undoubtedly be gaps where TLA requirements are not met by any existing standard. As discussed in section 3.2, a proving ground will be needed where emerging ideas can be explored, evaluated, and promoted into community-curated specifications. It is important to ensure that new ideas forged in the TLA development lifecycle give back to the community, and it is equally important that the community has a viable avenue for contributing to TLA requirements gathering and specification development.

3.3.2 Easy to Use
It will be critical for gaining momentum in the community that the TLA is easy to understand and easy to use. Of course, when we say "use" we then have to identify the user. In this case, the user is not the learner, but the personnel necessary for creating a TLA instantiation. There are several user types that need to be considered including:

- Organizational level decision makers
- Instructors and course creators
- Developers
- Network/IA Administrators

Instantiation of a TLA instance requires collaboration among all types of users, and typically the contributors will not all be part of the same company/organization.

There are two very basic principles that are a good first step. First, the TLA specifications should be human readable and clearly written to be meaningful to a wide reading audience. The specification must clearly describe what the value proposition is for participating providers. It must also clearly describe how and when to use the different parts of the architecture. In Year 1, valuable feedback was gathered from the experimental prototype development team indicating that for Activity developers it was difficult to understand how the output components they developed influenced the behavior of other components in the TLA (that consumed the output they produced). The TLA Specification constitutes a contract that facilitates component-to-component communication in software, so we will need to improve the description of how inputs and outputs function together. To make it easy to make the leap from a developer understanding the specification to a software system using the specification, the specification format should also be machine interpretable. If a computer can directly "read" the specification, then the developer has less work to do; there is no intermediary step of converting a human-readable format into a machine-interpretable format.

For a TLA ecosystem to be successful, many Activity providers will need to contribute individual activities so that collectively there are many available for learners to benefit from. Imagine how boring the internet would be if individuals from all over the world didn't feel motivated to add content for the world to see. The principle is the same for the TLA: the specification should make it easy for those who have content to share, to share it. And it is not only learner-facing content we want to encourage providers to share. We want smart people and organizations who have the ability to build
computational components that can reside inside the TLA Cloud to be able to contribute those offerings with equal ease. The key takeaway from Year 1's experimental prototype was that "plugging in" a new component into a TLA instantiation was too hard/manual/time-consuming. This bears surprising similarity to the "tyranny of numbers" problem faced during the invention of the microchip (see https://en.wikipedia.org/wiki/Tyranny_of_numbers). The basic premise was that it was possible to envision a hardware system that was more complex than it was cost-effective to wire together by hand. The eventual solution was automation leading to microchips as we know them today. We are at a point in history where this is now true of software: we can envision systems significantly more complex than it is cost-effective to build, due to the manual nature of connecting software components together by hand to meet the needs of individual use cases. The TLA's goals put it into this category: we want an ecosystem where it is easy to "plug in" components and reconfigure an instantiation to meet the needs of newly discovered use cases. To make it easy for developers to do this, new advances in automation will be needed to solve some aspects of the problem. For example, course creators are accustomed to manually sequencing content, but if the TLA succeeds there will be millions of pieces of available content for course creators to sift through. How can they do so effectively? The answer is certainly some form of automation, but what kind is not yet clear. For areas such as this, the TLA specifications must be flexible enough to allow the insertion of such an answer when it is discovered in future experimental prototyping. However, some aspects of simplifying "plugging in" already have industry-accepted solutions. One such example is automated component discovery. Automated component discovery allows components that "plug in" to register themselves when they "plug in" and then describe themselves to other components that are already plugged in. The TLA specification needs to include support for automated component discovery to make it easy for developers to contribute components to the TLA ecosystem, and to locate and utilize components provided by other contributors.

Instantiating a software system requires resources, computing power, disk space, and network connectivity, all which must be purchased. To make it an easy decision for decision makers and IT personnel to endorse the TLA, the TLA specifications must help alleviate their cost concerns. Our Spring 2017 demonstration at Fort Bragg utilized Amazon Web Services (AWS) for acquiring our computing resources. We saw first-hand how success at enticing participants to contribute new and unexpected components (in this case real-time dashboards), could lead to an unexpected spike in the usage of AWS resources, which has a direct monetary impact. To facilitate the balance between creativity and cost, the TLA specification must encourage implementations that are good denizens, respecting the cost of computing and storage resources. The specification should make it easy for components to publish their computing resource needs and negotiate for a reasonable allowance when they are "plugged in." Additionally, the TLA specification must allow the implementer the choice of placing their computational resources in the cloud (e.g., AWS) if they desire. However, this should not be mandated; the TLA specification should also be compatible with implementations that utilize a more traditional server-based installation within a single organization. In addition, the TLA specification must enable cost-based, real-time throttling so that IT Administrators can throttle components that are using more than their fair share of resources to prevent cost overruns.

Because the TLA is intended to be instantiated and left running for long periods of time, potentially many years, performance and maintenance needs must be considered. For a smooth learner experience the TLA should always appear responsive. To make this easy to achieve, the TLA specification needs to include the ability for components to publish their quality of service (QoS) requirements for input they
require, as well as the QoS levels they can support for the output they produce. By allowing communication of these important aspects in the specification, it is possible to assemble instantiations that meet both real-time and non-real-time use cases. Similarly, components can fail over time. For a continuous, smooth end-user experience, the TLA specification must enable fault tolerance. For example, hot swapping must be supported so that a failed component can be recognized and replaced without compromising system availability.

Another ramification of TLA instantiations that have life spans numbered in years is that large quantities of data will be accumulated. This is very desirable from a researcher perspective. However, there are practical matters to consider regarding data lifespan. As discussed above, throttling may be necessary to prevent cost overruns in storage by limiting data accumulation. Additionally, the TLA specification needs to facilitate data removal. This might be necessary to limit historical data to a life-span that is shorter than the lifespan of the TLA itself. Protection of personal privacy is another example of when data deletion may be needed (see PS4TLA: Privacy Support for the Total Learning Architecture Specification Document, version 0.1 for more information on privacy). An individual learner may wish to remove their profile entirely and leave the TLA ecosystem for reasons of personal privacy. To support this, the TLA specification would need to allow deletion of data that is associated with that individual learner.

3.4 DATA CENTRIC
The goal of the TLA is to improve the experience for the learner, and, thus, improve overall learning. One of the key outcomes of Year 1’s design-based research was realizing the extent to which this boils down to facilitating data exchanges. Rather than keeping data relevant to individual learners locked away in individual systems where it is only accessible from within the system that created it, the TLA aims to facilitate sharing data across learning systems. By opening up communication to allow data from one provider to be accessed by another provider, a more complete picture of learning emerges that enables smart components to produce better behavior because they have more information about the learner to work with. In this section, we will explore some of the dimensions required to encourage data sharing.

3.4.1 Lots of Data
It became apparent over the course of Year 1’s design-based research that to realize its full potential, the TLA needs to increase the amount of data that TLA participants can share. At the beginning of prototyping in Year 1, we put forth an initial, minimal set of data to share with the hypothesis that it was sufficient for the April 2017 Fort Bragg prototype. What we learned over the course of constructing the prototype was that many more kinds of data were desired by participant developers than we initially anticipated. And further, some of the data was not just desired, it was required to make the prototype functional. Some of the types of additional data included more xAPI verbs, more learner profile fields, more learner context, and more detailed descriptions of activities and competency alignments. The takeaway lesson was the more data we can share, the better!

As we explored the data exchanges necessary to make the April 2017 experimental prototype functional, we learned that the data shared by the TLA ecosystem covers many functional areas. It was difficult in the preliminary TLA specifications (version 0.1) to convey how and when to use each type of shared data, in part because it was unclear how to categorize the data. The preliminary categorization attempt was component centric, with sets of data organized by the components that would store the
shared data. This approach proved sub-optimal for two reasons. First, there were already too many data models; it was confusing to try to keep them all straight. Second, a component-centric categorization does not scale well as use cases are added. As described in section 3.1.4, different use cases require different sets of instantiated components. Therefore, categorizing the data models around a use-case centric aspect will result in an explosion of data models as more use cases are added. Analyzing the types of data that Year 1 has elicited as valuable has revealed that a simpler categorization is possible. All TLA data can be divided into three categorizations that are applicable across use cases:

1. **Learner**: Data that is about a specific, individual learner
2. **Science of Learning**: Data describing how humans learn
3. **Asset**: Data describing the characteristics of specific software components

Therefore, to be effective, the TLA specifications should have a small number of data models that are easy to understand, yet capable of sharing large quantities of data such as the three presented here. To scale effectively, the TLA specifications should use a non-component centric data organization scheme, so that it is not necessary to add new data models each time a new use case is identified.

### 3.4.2 Plug and Play

As anticipated, desire for many more types of data describing individual learners was elicited throughout Year 1, ranging from demographics data to certifications to preferences. While we anticipated data about the learner to be a primary area for expansion, it was surprising how much data about software is needed! For example, Recommender software components (henceforth called Recommenders) need to know about the available Activities in order to make good recommendations. It is not enough to know what a learner has experienced already and what they need to improve in. To make a good recommendation, the Recommender also needs to be able to consume data about potential Activities the learner could perform. Similarly, for a Recommender to present a remediation or challenge recommendation, it must have access to relationships between competencies and various science of learning approaches to sequencing content. Asset and science of learning metadata is critical for enabling smart software components to perform their algorithms effectively. In addition to enabling smart components, the asset and science of learning metadata also helps researchers and course curators. With this data available, the software architecture can allow multiple underlying learning theories to be "plugged in" so that researchers test and refine them. Similarly, it also supports "plugging in" multiple theories for sequencing a learner's trajectory through available content so that course curators can choose what best suits their educational agendas. Therefore, the TLA specification needs to provide a location for specifying metadata about software and science of learning methodologies to enable "plugging in" software components that provide differing approaches to learning.

### 3.4.3 Producers and Consumers

Independent of the specific transport protocol that is employed, our requirements analysis has revealed that all TLA data sharing can be described using a producer/consumer data flow model. All types of data either used in the experimental prototype, or requested for future inclusion, have two properties in common. First, each datum is the output of some component, or data producer, in the TLA ecosystem. Second, each datum produced can be valuable as input to another component, or data consumer. The data producer does not necessarily need to know what the data consumer will do with the data. However, for the data consumer to make effective use of the produced data, some amount of metadata describing how to properly interpret the produced data is helpful. For example, let's consider an Activity
that produces a "passed" xAPI statement. Without knowing which competency the "passed" statement is associated with, it is difficult for an Evidence Mapper to know how to interpret the meaning of "passed" in relation to competencies. But if the Activity also provides metadata indicating the competency that the "passed" statement is aligned with, then an Evidence Mapper can make an inference about the learner's mastery level in the associated competency. While the Evidence Mapper as the data consumer needs the competency mapping, the Activity, or data producer, does not need to know how the data they publish will be used. This is highly desirable, as it fosters creativity, encouraging new data consumers to find new and novel ways to use the learning data to benefit the learner. To facilitate a producer/consumer data flow, the TLA specifications should clearly describe:

- Data Concept: What data is shared?
- Data Structure: How is the shared data structured?
- Conceptual API: What can be said about the data between producers and consumers?

The term Conceptual API is used to distinguish the logical data exchange that takes place from the over-the-wire transport protocol used to send the data. The conceptual API is concerned only with what can be said, not how to send it over the wire.

3.4.4 Extensible

Because the TLA is designed to encourage new use cases (as discussed in section 3.1), it is anticipated that the data that will be exchanged using the TLA will grow over time. How does this impact the specification? It means the TLA specifications must provide a mechanism for adding new data required by new use cases, without requiring a specification modification each time a new piece of data is desired. Additionally, we have established that TLA communication between producers and consumers is centered on exchanging data. So naturally, if the data itself is extensible, then what can be said about the data must also be extensible. To accomplish this, the TLA specification should specify data exchanges in a general way that will be flexible enough to facilitate applying the TLA to new and unexpected use cases without requiring a specification modification each time a new use case is introduced.

API specifications, in general, describe data exchanges and manipulation. Existing paradigms for how to structure API specifications range from CRUD (create/read/update/delete) where the interface is strictly limited to simple data storage operations that do not vary with the data type or use case, to extensive APIs made by individual vendors that are tightly coupled to both use cases and data such as the Windows API (https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx). Because the TLA must accommodate new and unexpected use cases, the TLA specifications cannot include use-case-specific assumptions. However, the design-based research of Year 1 indicates that more than simple CRUD operations are desired. For example, let’s consider the use of Nuxeo in the 2017 Fort Bragg experiment. Nuxeo was used as a stand-in Activity Index because it already existed and could be repurposed to approximate Activity Index behavior without a large coding investment (valuable as Nuxeo is not intended to be a long-term solution for an Activity Index). One of the key developer complaints about Nuxeo in Year 1 was its clunky query API. The data was available, and it was possible to retrieve, but the API for retrieving it was sufficiently difficult to use that developers avoided using it, instead hand-synchronizing duplicated data sets internally. Therefore, to be effective, the TLA specifications need to provide a developer-friendly API for data consumers to use to acquire and filter data a producer can provide. In particular, some data consumers will want all the data that a particular
producer can provide, whereas other data consumers will be more selective and will want to exchange only small amounts of data.

3.4.5 Real-time Data Processing and Stored Data History
At the onset of Year 1, it was not clear how much of the data exchanged between producers and consumers should be stored. Would it be better to process the data on the fly in the live system and not store it at all? The 2017 Fort Bragg experiment was instrumental in identifying an answer. While the technology does exist to process big data in real-time, this does not fully meet the needs of our preliminary use case (meta-adaptation; see TLA Use Cases.docx for more information). During the experiment, several data display dashboards were created that could display learner progress information in different ways in real-time. The dashboards were kept up and running live during the experiment for long periods of time. In addition to the anticipated use of seeing a live snapshot of "right now" in the system, timeline style charts that the dashboards created also displayed a visual history. The visual history aspect was of just as much interest to instructors, learners, and stakeholders as the moment to moment snapshot. However, because some of the dashboard prototypes were not storing any history after they exited, as soon as the dashboard was closed, the historical view was gone. Similarly, if the dashboard was not up and running when an interesting event occurred in the system, that event could not be visualized afterward if the data about the event was not stored. Thus, one outcome of the experiment was that TLA specification must support storing data, or history, to support offline data analysis needs.

Within the single meta-adaptation use case, the experimental prototype also demonstrated that the TLA specification needs to support real-time data exchanges as well as non-real-time data exchange. For example, if the learner completes an Activity, then the completion could trigger an inference about the learner's mastery estimate in a related competency. The mastery estimate update might, in turn, cause a Recommender to suggest the next most appropriate Activity. Ideally, the learner should not have to wait a noticeable amount of time for this sequence of events to occur. Their perception should be that they completed one Activity, and immediately have a recommendation for the next. Therefore, the TLA specification of what the data is and how data producers and consumers exchange it must be compatible with both offline and real-time processing needs.

3.4.6 Distributed Data Models
We must also consider where the data that is stored should reside. We described three logical data models (learner, science of learning, asset) in section 3.4.1. However, the logical categorization does not necessarily need to map one-to-one with a backing database. Let's consider two elements of the learner data model: the master estimate describing the learner's ability in a specific competency, and the learning goalsthat describe which competencies the learner needs to learn. Logically, both are parts of the learner data model. However, physically, these different data elements might reside in different components instantiated in different physical locations. The TLA specification should support a distributed data model, where the data can reside in different physical locations, but be accessed by all data consumers as though it were a single, globally shared data model. That is, while the goals and mastery estimate might be provided by different components in different locations, they both can be considered as sub-sections of the logical learner data model. Arranging the TLA specification in such a way allows different providers to add new aspects to the learner data model without requiring the data to be transferred to a physically centralized data storage location. This is highly desirable as it will
encourage better community collaboration. To achieve this goal, the TLA specification needs to be able to support descriptive metadata so that data producers can specify what parts of the logical data models they produce, and decide whether or not they will store the data they produce for later retrieval.

3.4.7 Performance Minded
Scalability demands that the specification approach consider performance as well. We have considered dividing the logical data models physically to allow collaboration between performers. Given that historical data is desired, and will grow over time, it is logical that supporting scale-out NAS techniques will also be necessary to provide the storage needed for the expanding historical records. The amount of data stored, the rate of data accumulation, and the demand for data retrieval from data stores all impact the processing power and storage space required to meet the demand. As the demand goes up, load balancers will need to be introduced to ensure consistent performance. The TLA specification of the data should not restrict or prohibit scale out and load balancing techniques, nor should it mandate them. Instead it should seek to be compatible with the latest state of the art techniques in load balancing and data storage techniques without requiring a specification modification to achieve performance goals.

In addition to the storage space and processing power required to manage the stored data, network resources must also be considered. As an example, some data consumers might wish to ask an LRS for all xAPI statements describing the experiences of a specific learner, whereas others might want to know only the last xAPI statement about a specific learner. Imagine if the only way to retrieve information from an LRS is to ask for all xAPI statements that exist. Over time, as the number of stored xAPI statements grows, the size of the response to "give me all xAPI statements" grows accordingly. Certainly, the data consumer could retrieve them all, then filter them as they like. But doing this has a performance impact. If the data consumer is doing the filtering, then all the data to be filtered must first be transmitted across the network from the data producer to the data consumer. Among other performance concerns, this unnecessarily wastes network bandwidth. To alleviate this problem, the TLA specification should describe what can be said in data exchanges about the data in a way that is powerful enough to enable filtering to happen at the data producer. This will make it possible for data consumers to ask more precise questions about the data such as "give me the last xAPI statement about a specific learner " so that they are respectful of shared network resources.

Additional performance aspects such as fault tolerance and throttling have already been covered in section 3.3.2. The TLA specification of the data models and their APIs must be compatible with the performance characteristics that section describes.

3.4.8 Automation Friendly
During the 2017 Fort Bragg experiment, one of the pain points encountered was the need to manually enter metadata about each Activity, which included manual entry of the competencies associated with the activity. Manually entering this information is subject to human error, and is very time consuming. While it was tractable to perform in the context of a small experiment, a manual solution does not scale very well. The observation from the design-based research in Year 1 is that if only a manual solution is available, it is very likely that the metadata is highly likely to be incomplete. Activities with insufficient metadata cannot be used effectively by other TLA participants, since there may not be enough machine-readable data describing them to determine how to utilize them. To address this difficulty, the TLA specifications need to provide programmatic means for adding and modifying metadata in a way that
makes it easy to automate data entry. Automation can range from simply shifting the data entry to a developer task (e.g., allowing smart content to programmatically report their own characteristics) all the way to plugging in smart algorithms to populate metadata. The TLA specification should be compatible with this wide range of automation techniques.

3.4.9 Security
Similar to the discussion of transport security in section 3.1.3, security of the data at rest must also be considered. That is, stored history, as discussed in section 3.4.5, needs to be stored safely. As with transport security, the at-rest security should be decoupled from the description of what the data is in the TLA specification. Doing so allows new security techniques to be incorporated as they emerge without requiring a change to the TLA specification.