The information contained in this document represents the current view of Stone Bond Technologies in the issue discussed as of the date of publication.

This white paper is for information purposes only.
Stone Bond Technologies may have patents, patent applications, trademark, copyright or other intellectual property rights covering the subject matter of this document. Except as expressly provided in any written license agreement from Stone Bond Technologies, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or intellectual property.

Stone Bond Technologies, LP
1020 Main Street Suite 1550
Houston, TX 77002
713-622-8798

What is Agile Enterprise Integration Software?

We hear so much these days about agile enterprises and agility in software, but what does that mean? An agile enterprise responds quickly to unexpected opportunities, buys companies and spins some off. It changes the product line to accommodate surprise trends, and can change its business processes almost overnight. While the spirit may be agile, the constraining element is invariably the underlying IT infrastructure's ballast. The word "ballast" is defined as: a heavy material used to enhance stability; that was definitely something important in the late 1990's and early 2000s when companies needed to consolidate their disperse infrastructures and applications. The integration architectures that are prevalent today came from that era, but the ballast has outlived its usefulness. Organizations simply cannot be responsive when changes in the integration infrastructure are measured in months and years rather than minutes and days.

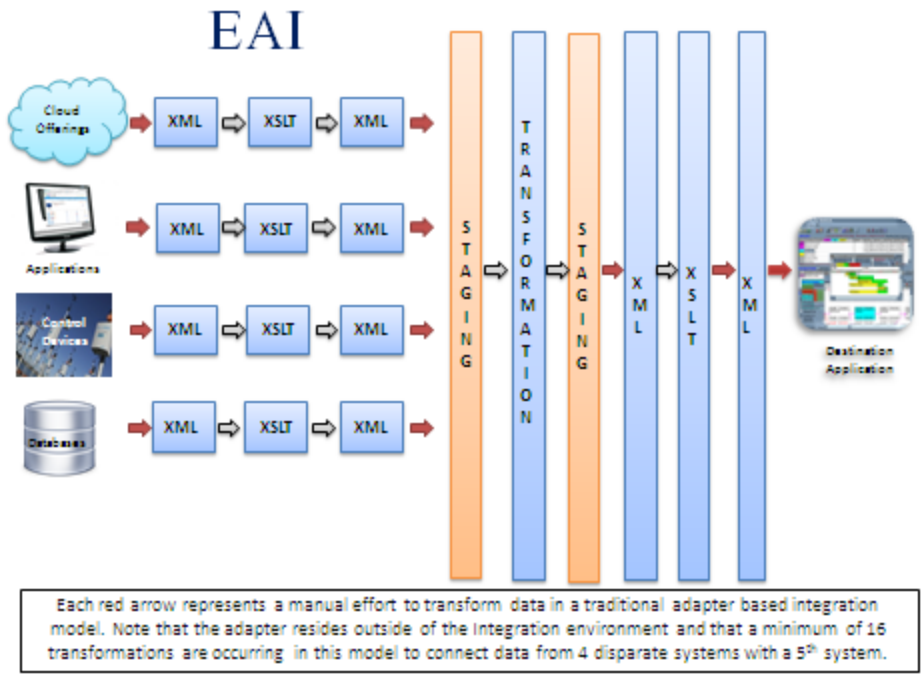
How are these companies going to keep up with competitors that were invented after the internet grew up? They must be able to leverage their proven experience and valuable existing applications on a new playing field, and the way to do it is to replace the ballast with Agile Integration Software [AIS].

This paper discusses ten key elements that can help you identify an AIS. Some of these points will be found in special purpose platforms such as standards-based B2B like EDI, limited scope integration around a particular endpoint like SAP, or SOA platforms. Remember, though, that we are looking for Enterprise level software that suits more than 90% of the integration needs of the enterprise, and allows it to keep up with and even outrun the youthful new competitors. You will find that all ten of these characteristics must be present and working together, in order to deliver the agility your organization needs to be competitive.

There are implicit requirements not included in this list that would make the software Enterprise ready, such as being able configure load balanced on clusters and the ability to connect to virtually any applications, data bases, and instruments.

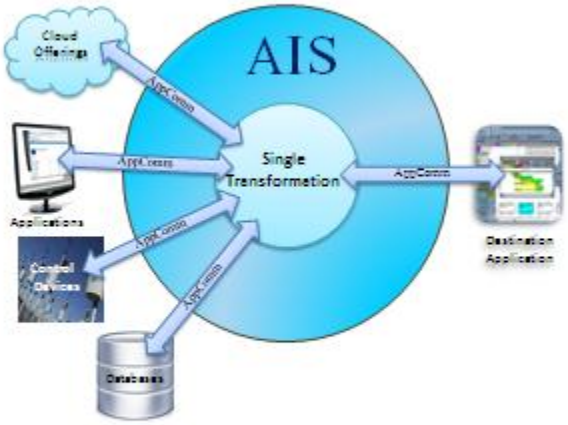
With EAI and ETL, there are many steps involving custom programming activities that are done over and over again, each time an interface is built. (See Picture 1)

Picture 1



The most fundamental philosophy in the design and development of an AIS is that anything done more than once should be automated. The software must amply but judiciously apply intelligent inference during the configuration of integration, and it must streamline all aspects of execution at run time. It must recognize that there will always be requirements that fall outside that realm, and for those, it must make powerful tools available from within the environment, along with the ability to extend the environment as needed. It must scale up for enterprise use but also scale down to bring the same power to any project or application. (See Picture 2)

Picture 2



Agile Integration Software is a homogenous ecosystem where all parts are fully aware of each other. AppComms live inside the same system as the transformation engine and all data is accessed natively, converted real time on the fly and delivered natively. In short, 16 transformations become 1.

1. "Off-the-Shelf"

While "off-the-shelf" may not always be explicitly the case, it can be used as a reasonably good "rule of thumb." If it takes days or weeks to install, that's a good indication that the software has multiple discreet components that most likely do not work as a cohesive holistic solution, one that would not be able to meet the next nine qualifications. After all, this is about agility. Fast implementation and configuration of the environment means the ability to respond to corporate change more quickly. Long, slow installation processes, by themselves, negate the concept of agility. Upgrades and updates for AIS are automated, as opposed to requiring laborious and repetitive manual efforts.

2. Virtual Relationships

This feature is absolutely essential to agility. You must be able to define virtual relationships across dissimilar sources in seconds or a few minutes, and then define mapping to your destination quickly. While you design the relationships and mapping, the associated run-time components should be generated so that you can validate and test as you go. Even better, if you want these interfaces to be bi-directional, the necessary metadata and run-time components should also be generated for you. At run time, the data from multiple sources is accessed live, aligned, transformed, and sent, as specified, to the destination.

In comparison, a typical ETL software handles only one-to-one mapping of data. That means that whenever a destination requires data that is aligned or logically derived from more than one source, it is necessary to define a data model that incorporates all of the necessary data elements. This becomes a staging database from which the data is extracted, again transformed, and then sent to its destination. The mere existence of the staging database implies significant design and implementation efforts, and means that anything close to real-time data transfer is impossible.

With EAI the approach is more variable, but requires custom programming and/or considerable effort to define common views of the data involved. Most EAI systems rely on an XSLT transformation engine, which requires that all data it processes be XML. This means that all sources must first be converted to XML. The relationships across the XMLs are defined various ways; in the end, a time-consuming effort.

There have historically been a couple of approaches to resolve this. An earlier manifestation of virtual relationships is known as database federation, data federation, or data virtualization. More planning and design is necessary, and these virtual definitions are not easily changed. Data movement takes two steps, just like an ETL, but at run time; first data is read, transformed, and moved to the virtual data model, then it is accessed, transformed, and sent to the destination. Like any staging database, when more applications are added to the environment, the data model no longer fits cleanly. It eventually either needs complete redesign or is allowed to become an insupportable mélange, and tend to be treated as, or become, static solutions, even though they are virtualized.

In many cases, the very existence of a data "warehouse" is primarily for the purpose of supporting integration. One of the biggest benefits of virtual relationships is the total elimination of the requirement for a staging database or even an interim virtual format such as XML, or in the case of data virtualization, a common database schema. A very huge percentage of the time (and money) invested in enterprise integration, or even integration to feed a dashboard, is spent defining and designing a database schema that accommodate all data needed from all sources. This is a big consulting gig, whether it is a virtual schema for data virtualization or it becomes a physical database where the data is stored. Maintaining this is a terrific headache and cost over time, and certainly is a significant impediment to agility.

Another very important aspect of not staging the data is the ability to access the actual data live from any/all of the sources, as opposed to using stale data. And, of course, performance; how can the performance of accessing data from the sources possibly be optimal if the first step is always to get the data to a central view before it goes on to the destination?

Note: It is important to note that in cases where data is stored in a database for reasons other than simply to make the integration easier, that database or data warehouse would be treated as any other data source.

3. Embedded Code Editors and Compilers

It is inevitable that some mapping of data among sources will be more complex than can be done with logic that is built into any mapping user interface. Handling those situations will require *ad hoc* code fragments or pre-built analytics. AIS will have standard code editors built in that can be accessed from various places within the construction of integration. You will be able to compose small (or large) functions that can access and act on any information involved in the integration. Wherever it makes

sense to be able to include custom code, you enter the code editor and test it in place. As part of the solution you are building, AIS will manage the code, store it with the other mapping or process definitions ("metadata") and ensure that it gets assembled in the right place. For example, you may be mapping fields A and B to destination field C, and the built-in capabilities do not meet your requirements. You should be able to enter the custom code option, and build and test the code. When the transformation engine maps those fields, it will execute your equation.

Mapping with an EAI is limited to the functionality provided in the mapping tool. If complex or specialized equations or processing is necessary, it must be handled as custom code outside of the mapping tool. That means you must leave the mapper environment, go to your coding environment, write, compile and test the code, then either plug it into the mapper, or make sure it gets called before or after the transformation at run time. You'll find yourself spending a huge amount of time switching environments and managing the separate pieces of the solution.

This valuable feature eliminates need to invest the time and disruption to leave the AIS environment and go to a separate development environment to write code, compile, and test.

Extensibility of the development environment is also an important aspect of AIS, as it allows custom components to become accessible and selectable as reusable formulas, dynamic adapters, pre-and post-processors.

4. Design-time Validation

Since the development time of integration projects is such a huge component, AIS has many angles that help to reduce the time and effort. You should be able to see any integration object (metadata) you are building validated as you are building it, advising you of things that you have not completed, and errors you may have introduced. You should also be able to test it from the same environment.

For example, as you build a mapping of sources to destinations, you should be able to see live data at the endpoints to make sure you select the right fields, and see constant validation as you define relationships and formulas. Once you have finished the mapping, and no errors show up, you should be able to push a button and see the results of the mapping from live test systems. If something doesn't look right, you can simply fix it and try again.

The need to move among environments: design, development, and test is eliminated, along with the associated time and frustration.

5. Transformation Engine that Reaches out to Applications Directly

AIS cannot rely on XML based XSLT transformation engines. These require that the data, regardless of its native format, must be converted to XML before it passes through the transformation engine, and must be converted from the XML output to whatever the destination requires. This means effectively that there are two transformations that have to happen, one on either side of the XSLT conversion. Doing this means additional development and additional steps at run time, which will impact performance. It also means that the transformation engine simply cannot coordinate the processing of virtual relationships across disparate systems.

Often a database analyst who is really familiar with the data in a relational database gets disoriented when doing mapping from and to XML schemas that look nothing like the databases of their expertise. Of course, this introduces opportunity for errors and slows the implementation process.

AIS eliminates the classic adapter, which is the source of much re-programming, and replaces them with intelligent "dynamic adapters." Dynamic adapters, called AppComms, are able to discover the schema from the specific instance of an application as opposed to making assumptions. This way, custom fields and tables (aren't most applications customized?) are easily accessible without additional custom coding. AppComms are able to communicate with one another at run time to coordinate key relationships and other synchronization that must happen. They are orchestrated by the transformation engine. You may have a situation where the data required comes from an ERP, a spreadsheet, SaaS, and an electronic instrument. All four source AppComms work together to access the right data in native mode and pass it to the transformation engine at the right time to resolve cross-application virtual relationships. This is what makes it possible to perform live access, alignment, and transformation of data that is owned by multiple disparate applications without ever making a copy of the data.

6. Single Environment for Development, Testing, Deployment, and Monitoring

It is important to have a single environment within which to build, test, modify, and monitor an integration solution. AIS requires that a person configuring an integration be able to test end-to-end and deploy without leaving the environment in order to minimize the speed of implementation and to reduce maintenance costs. All the metadata that define the data access, business rules, and triggering events is generated and managed from that single User Interface.

Working with EAI and ETL means moving in and out of multiple environments to build - compile – test - correct - test – deploy. This not the case for an Agile Integration Software.
A huge benefit of an end-to-end environment is the ability to access virtually any information about the data, application, error status, etc, and to use it to build conditions, formulas, custom code, and such. At run-time, AIS supports sharing of data and metadata throughout the system. Imagine being able to pass run-time variables to formulas or instruction for the transformation engine to be able to change the way it operates based on data captured in a workflow or perhaps information from a non-participating application.

7. Data Workflow Logic

AIS must include the ability to define data workflow logic in the same environment and must make both the data logic and the workflow logic and information available for use throughout the AIS at run time. You will find the same philosophy of automating behind the scenes, embedding code, reusability, design time validation, and integrated testing. A separate tool for workflow will not support the fluid end-to-end information exchange at execution time.

EAI and ETL typically have a separate tool for data workflow logic.

8. Secure, Auditable Infrastructure

Discussion about security of an integration environment generally focus on security of data as it is being moved, and on ensuring correct end user permissions when accessing data that has been extracted from an application.

AIS adds another dimension to the infrastructure security capabilities. By eliminating the need for leaving the development environment to add custom code, and by having the execution centralized, AIS ensures a locked-down infrastructure where it is impossible to tamper with the underlying integration infrastructure by changing code, metadata, mapping, or other integration objects without being tracked. Every change that is made to any integration object has an audit trail of time stamped “who did what.” Users of the development system not only have granular permissions/restrictions granted with respect to activities, but also to system or process accessibility across the enterprise. This is critical in certain environments such as government and healthcare.

EAI and ETL integration infrastructure are open to rogue programmers who could easily make a change to divert or modify data streams.

9. Embeddable in Software Applications

Any ISV should be able to provide all of this capability to their applications and customers. Because of its compactness and ability to scale, not just up, but also down, AIS is embeddable or can be called from any application, so that ISVs can provide integration as part of their solution. EAI's cumbersome nature and time to implement, as well as its price point, make it impractical for ISVs to incorporate it into their offerings. Technically, EAI has no hope of being embeddable.

Some ISVs require their customers to perform the integration necessary to their backend systems or to populate a dedicated database for its use. Customers may use EAI, ETL, custom code, or whatever is familiar to them. Many times, this is such a prohibitive exercise that causes ISVs to lose sales. AIS, by its nature, is easy to embed or to package with an offering.

For example, AIS embedded in SharePoint 2010, provides full CRUD (Create, Read, Update and Delete) capabilities including cross-application virtual relationships. Any WSS 3.0 application or Web Part can get the same functionality via an AIS ADO.NET driver (Full CRUD). For older applications that use OLEDB, the AIS functionality is available read only with the OLEDB driver.

AIS can breathe new life into BPM implementations, providing bi-directional access to merged and aligned data from multiple back end systems without staging the data anywhere. Dashboards can reach out directly to the systems of record for live data at any time, and gain another dimension with bi-directional integration. Rather than just a means for viewing data, dashboards can also be used to capture and initiate decisions and data or updates that the end user makes based on the information.

10. Integration Infrastructure Change Management

Once an integration infrastructure is in place, the great challenge for corporate agility is rapidly incorporating changes without incurring significant risk and disruption. The imperative comes from mergers and acquisitions, changing business direction, or just an upgrade to a new version of software. The IT infrastructure has always been an impediment to the agile company because technologies for integrating software, data sources, and business partners cannot support or withstand much change without beginning to incur exponential instability.

Any time a change is made, AIS prevents the potential catastrophic side effects of crashing systems or bad decisions because of bad information.

AIS watches for changes and performs an impact analysis every time any change is detected in an application, data source, business process, or element of integration such as interfaces, business rules, etc. Once the potentially impacted elements are identified, the process notifies the owners of the objects with detailed information. The owner either approves the change or makes a change in his object(s) to accommodate, which would again start the integrity check, continuing recursively. All changes are held pending until completely resolved, and then are automatically rolled out in the appropriate order.

When changes are required in an EAI environment, you would usually plan long in advance and essentially start from the ground up.

AIS's Integration Integrity Process means that changes can be applied more quickly and with dramatically reduced risk of errors or adverse impact, so that the company can respond to changing market forces in an agile manner.

Conclusion

Companies adopting AIS will be positioned to reap the benefits of agility and the potential resultant competitive advantage. Huge integration projects will be reduced to establishment of ongoing improvement processes based upon agile integration software and principles.

If you are wondering if it is time for your company to invest in moving to AIS, you may want to ask these questions about your company:

1. Do you have any mergers or acquisitions planned?
2. Do you have any major software upgrades planned?
3. How many legacy applications is your business dependent upon?
4. Do you have SOA projects that need to incorporate non-compliant systems?
5. Do you have a Microsoft SharePoint community that could benefit from bi-directional secure live data from multiple applications?
6. Do you have KPI dashboards that are hampered by the static and brittle nature of the behind-the scenes integration?
7. Do you have specialized, niche applications that need data from other places, and suffer from poor integration?
8. Do you have SaaS that needs to exchange data freely with other SaaS or with your in-house applications?
9. Do your automated business processes need to trigger data flows behind the scenes?