Introduction

Abstract

In recent years, companies involved with both open source and proprietary integrated library systems (ILSs) have made a concerted effort to increase the openness—that is, the degree of flexibility and interoperability—that their products offer to librarians and programmers. This chapter of "Opening Up Library Systems through Web Services and SOA: Hype or Reality" explores this dynamic in the current ILS market. By taking a general look at what characteristics and functionality librarians seek in their software and defining key terms, this chapter sets the stage for an in-depth exploration of the modern ILS trend towards APIs, Web services, and the service-oriented architecture.

In the current phase of library automation, we've become inundated with the language of openness. Open source integrated library systems (ILSs) have emerged, promising to give libraries more control over their software than has been possible with proprietary, closed-source products. Companies that produce and provide service for proprietary products have redoubled their efforts to offer more flexibility, openness, and interoperability through Web services and other application programming interfaces (APIs). A new front has developed in the competition among library automation alternative vendors, who are racing to open up software and allow libraries more access to their data and internal functionality. This new emphasis on openness can be a great benefit to libraries to the extent it that it actually offers new capabilities otherwise not available. Still, as implied by the title of this issue, it's often difficult to distinguish products that fully embrace openness from those where the claims don't quite match reality.

In today's environment, systems that are perceived as being "closed" have diminished appeal. It sure sounds better to characterize an automation system as "open" and flexible, but what do the terms really mean? We will explore some of the techniques that provide increased access to data and internal functionality, focusing especially on Web services and other application programming interfaces.

Many libraries might say that they do not want a "black box" system that restricts users to the functionality in the interfaces provided by the vendor, with no access to the internals of the system. Yet many libraries need a turnkey system that helps them carry out their work without the need for any local programming or intervention. We should emphasize that APIs offer additional opportunities for those that want to do more with their software, but do not impose any technical requirements for libraries that choose not to use them.

When it comes to the purveyors of proprietary software, claims of openness are also everywhere. The emphasis on openness may have been accelerated by the open source movement, but it has been a steady theme for many years. Press releases and product literature gush with the language of openness. The means to this openness are the adherence to standards and Web services and other application programming interfaces.

This report aims to take a close look at the major ILS products on the market and describe the approach that each offers in delivering open access to its data and functionality. Of particular interest are the APIs that each system offers to the libraries using its product. We will describe and evaluate their scope and comprehensiveness and observe the extent to which each product offers these APIs through Web services, the preferred approach in the current phase of information technologies.

Libraries have varying expectations when it comes to what they want from their automation products. Many simply want the system to function as documented, using the interfaces and reports delivered with the system. Some ILS products target libraries that expect basic functionality without the expectation of local programming. The products in this category serve a vital role in the library automation arena. Libraries expecting to work with their systems through local programming should be clear on which products offer this capability and which do not.

Scope

This report focuses on integrated library systems, the core business application used to automate the work that takes place in libraries and to provide access to their collections and services. Libraries make use of many different software components, such as discovery interfaces, link resolvers, federated search tools, digital collection management systems, and institutional repository platforms. Although many of the same questions apply to other genres of software, those lie outside the scope of this report.

We further narrow the scope to the ILS products widely used in the United States by the kinds of libraries likely to have an interest in extensible systems. The types of libraries in this category would include large research libraries, municipal and large public library systems, and national libraries. The specific systems examined will include those from Ex Libris, Innovative Interfaces, SirsiDynix, VTLS, Polaris, and The Library Corporation, as well as Koha and Evergreen, which are widely adopted open source ILS products. Even though Talis does not market its ILS in the United States, it has been an active proponent of this approach and warrants inclusion.

Intended Audience of This Report

This report aims to provide useful information to anyone involved with automation products in a library context, especially those involved with defining or implementing technology strategies. Library administrators will find information, presented in clear, nontechnical language, that will help them understand some of the options to extend and enhance their automation environment. These extensions relate to extracting and manipulating data in ways that will support management decisions, allowing the library's computer systems to work together more efficiently and to better connect with the information systems of the broader organization. For library administrators and other nontechnical professionals, this report will provide information and context to help understand the claims and counterclaims of open source and proprietary software proponents.

Developers and other technical staff may already be familiar with many of the concepts explained in this report, but should find the examinations of how different APIs have been implemented in particular products to be of interest. Systems librarians, Web developers, and others ready to extend their activities to include programming with the library's ILS or other key infrastructure components will learn basic concepts and the realm of tasks that can be accomplished using these interfaces. Developers outside the library industry who may be involved with libraries will find important information on the integration capabilities offered by the major library automation products.

Librarians and other library workers not directly involved with technology will benefit from understanding the concepts involved since this is an area of technology that has a direct impact on the information environment of the library and the information and functionality that supports their work.

The ability to work with library automation software through an API benefits some types of libraries more than others. Those involved with larger and more complex library organizations will have more opportunities to take advantage of the APIs and other integration technologies covered in this report. The key target organizations include academic libraries of all sizes, large and medium-sized public libraries, special libraries supporting large organizations, and national libraries. These libraries generally operate a variety of information systems within their enterprise networks, with interdependencies that often cannot be realized by out-of-the-box functionality. The libraries require custom programming to get what they need out of their software. Smaller libraries tend to use automation products as delivered by their developers, increasingly as a hosted service, and may have fewer needs that require programmatically extending the functionality. Smaller libraries are also less likely to have the technical staff to implement these capabilities.

Why Should Libraries Care about Application Programming Interfaces?

The integrated library system, or ILS, provides the essential automation infrastructure for a modern library and represents one of the largest technology-related investments that a modern library makes. Libraries select from a variety of major products on the market, including both proprietary and open source flavors.

Each library brings a unique set of expectations and requirements to the table as it implements its ILS. Through a careful selection process, the library will identify the system best suited to its fundamental requirements. Yet no prepackaged automation system will completely satisfy

all of the nuanced needs of every library. Equipped with an API, libraries with their own programmers have the option of creating functionality that fills in the gaps between the system as delivered and their specialized requirements.

Library automation systems increasingly operate within a broader context of information systems. Especially in larger organizations, the ILS needs to communicate with a variety of other business applications.

Any ILS comes as a complete package designed to present a broad set of features and functionality through the user interfaces delivered with the system. These user interfaces, whether presented through a Web browser or implemented through a graphical Windows, Java, or Macintosh interface, allow human users to interact with the software, searching for resources, performing transactions, extracting reports, or carrying out other business functions. In selecting a system, a library measures the completeness of that system in terms of what can be accomplished through these user interfaces.

The user interfaces provided with the ILS are the products of the people who develop the system. While some aspects of the user interface can be adjusted by the configuration options selected by the library, the basic functional capabilities cannot be altered except by those involved in the development of the software.

While many libraries find the functionality delivered with the ILS to meet their automation needs, others benefit from the ability to perform tasks beyond that of the delivered software. A robust and well-documented set of APIs empower the library to perform tasks with the ILS and the data it manages that go beyond the delivered system.

APIs associated with an integrated library system enable interoperability, make its functionality extensible, and empower the library to be more independent of the organization that created the software:

> ***Interoperability.*** For many libraries, a key concern lies in their ability to make the ILS communicate effectively with other computer systems. The more that a library exists within an organization that makes use of multiple information systems, the more that it needs an ILS that can interact with those other systems. In such a context, an ILS that cannot interoperate with other systems functions as an isolated silo that may not support the library's organizational and business needs. To a large degree, interoperability can be achieved through adherence to applicable national and international standards. Yet standards do not necessarily address every possible aspect of the way that a library might need its ILS to interact with other business or information systems. APIs pick up where standards leave off, allowing libraries to create interoperability that cannot be achieved in other ways.
>
> ***Extensible.*** An ILS embodies a specific set of features and functions that are needed to automate the internal operations of a library and provide its users with access to its collections and services. The set of features in a given ILS will continue to evolve over time in response to the ongoing enhancement requests that emerge from the libraries that use the software and from the development agenda of its developers. Many libraries have specific automation needs not fulfilled by their ILS. These needs may not be shared by other libraries that use the software, making them unlikely candidates for the normal enhancement process. An extensible system allows the ILS to enhance its functionality without the intervention of the programmers that created it. APIs provide one of the most important vehicles for extending an ILS according to the needs of a given library.
>
> ***Vendor independence.*** The presence of a robust API can help reduce a library's dependence on the organization that created and supports the ILS. With a closed system, only the vendor can make changes to the system to extend its functionality. An API provides a library with the ability to create customized functionality without the intervention of its developers. This capability enables the library to have more flexibility; it is less hampered by an unresponsive vendor and can accomplish tasks with its own staff that otherwise might require paid custom programming from the vendor.
>
> The degree of independence gained by the ability to take advantage of an API isn't absolute. The library continues to be dependent on the product's developers to maintain the product. If the company goes out of business or withdraws the product, the programming invested may or may not be transferable to another ILS.

Possibilities Abound

To help readers visualize why it's important for an ILS to offer an API, some of the tasks that can be accomplished might include:

> ***OPAC replacement or enhancement.*** One of the ma-jor trends today involves the transition from traditional online catalogs to new-generation discovery interfaces. These new products often come from sources other than the developer of the ILS, but thoroughly rely on the ability to extract data from and communicate in real time with the ILS. It's through APIs that it becomes possible for a third-party discovery interface to work seamlessly with the ILS.

Many libraries need to enhance their existing online catalog in ways beyond the standard configurations that are offered. The display of book jackets, summaries, or reviews can be layered onto catalog pages using an API.

It's of increasing interest to embed library content and services in external Web pages and portals. A flexible set of APIs in the ILS, especially in the form of Web services, will help support these capabilities.

Connectivity with self-check and automated materials handling equipment. The ability to take advantage of self-service and other external automation systems may be enhanced through APIs. Much of this connectivity is addressed in SIP2 and NCIP protocols, but libraries may also be able to enable additional efficiencies if they are able to supplement these protocols through other APIs.

Single sign-on and authentication services. A key problem that many libraries face involves how their users sign in to their various applications. Given the multitude of systems, it's important to have some means of consolidating the function that controls how users sign in. It may involve configuring the ILS to rely on an external authentication service, or it may mean the ILS functioning as an authentication service for external application. Either way, the ILS must support the APIs associated with authentication, such as LDAP, ActiveDirectory, Kerberos, or Athens.

Financial system integration. Many libraries need to be able to exchange financial information with other business systems in their institution. Procurement and fund management tasks that take place in the acquisitions module of the ILS often need to be transferred to an enterprise resource planning (ERP) program (a genre of software that large organizations use to manage their financial data across different units) or other accounting systems for payment. In academic libraries, fees incurred in the library may need to be transferred to a bursar's office for collection through the student's institutional account. Some libraries may want to offer online payments though through their own or though their institution's site using third-party e-commerce products.

Detailed reporting. Although all ILS products come with reporting modules that generally include the ability to produce customized reports, they may not have the ability to address all aspects of data managed within the ILS. Many libraries are able to use an API to extract data in ways not possible through standard reports.

These tasks represent only a few of the possibilities. Equipped with a well-developed API, a library should be able to respond to a wide variety of needs that arise involving some aspect of information managed within its ILS.

Basic Concepts

This report focuses on techniques for providing libraries more open access to their core automation systems. The key approach that we will explore is the application programming interface, or API. In today's technology environment, the preferred implementation of an API is through Web services, which takes advantage of the protocols, structures, and technologies that support the Web. Systems formed entirely out of Web services and that follow a particular set of organizational principles can be said to follow a service-oriented architecture, or SOA.

One of the most important concepts to understand about an API is that it involves computer-to-computer interactions. Not to be confused with the user interface offered for humans, the applications programming interface involves allowing one computer system to interact with other computer systems. As an application *programming* interface, it involves programmers. Those libraries lacking technical staff capable of at least some software programming will not work with the APIs directly. Libraries without programming staff may benefit indirectly through capabilities executed on their behalf by third parties.

Some systems may have internal APIs designed for the developers of the system, but these APIs may not be packaged in a way that makes them accessible to the personnel in the libraries that use the system. The presence of APIs within the internal system framework designed for use by the software engineers that program the application itself may not be palatable platforms that will help the personnel in the libraries that use the software customize it and maximize its capability. While the use of internal APIs reflects modern software design, we're primarily concerned with APIs designed to be used by the libraries that implement the software. It's these customer-facing APIs that directly benefit libraries using the software, more so than the APIs that are geared more for use by those involved with developing the application itself.

A customer-facing API needs to be largely abstract from the internal workings of the underlying system. It should offer high-level operations that do not require detailed knowledge of the internal programming conventions within the ILS. The API should be independent of any given programming language or operating system. Most importantly, it must come with detailed documentation that provides the library programmer with adequate information on the requests supported by the API, the protocols and syntax involved, and the form of the expected response.

API Implementation Models

As we approach the realm of application programming interfaces, let's think of applications that lack this architecture and step our way through progressive levels of support. Figure 1 illustrates the most closed approach to programming. This type of application is completely self-enclosed. The software has exclusive access to any of the data involved. Whether the databases used are proprietary or based on standard relational database management systems (RDBMS) products, the customer has no access to them other than through the interfaces delivered as part of the software application. All of the functions of the software likewise cannot be accessed except through the interfaces delivered with the application for staff access or public access, or through a reports module.

Whether or not an application offers an API has little to do with its completeness of functionality, sophistication, or scalability. While ILS products that target small libraries tend to be more self-enclosed, some of the legacy systems serving larger libraries might also fall into this category. The point of distinction here involves the nature of the application as entirely self-enclosed with no programmatic access to its internal data or functionality.

The extent to which a system offers APIs relates, to a certain extent, to its vintage. Products designed more than five years ago may not have originally incorporated APIs in their design. As these products evolved, most have come to include APIs that expose functionality through a modern programming interface that may depend on internal proprietary programming.

Any major software application developed today would be programmed in a way that would naturally involve APIs. The prevailing preferred approach to software development involves the service-oriented architecture (SOA), which fundamentally embraces the concept of APIs in the form of reusable services. These services implement small units of work that form the building blocks of larger applications. The services may be used within the application at hand or exposed externally. Although any new automation system created today would surely be service-oriented, most applications available in the library arena predate the emergence of this approach and have to be retrofitted with service layers or APIs.

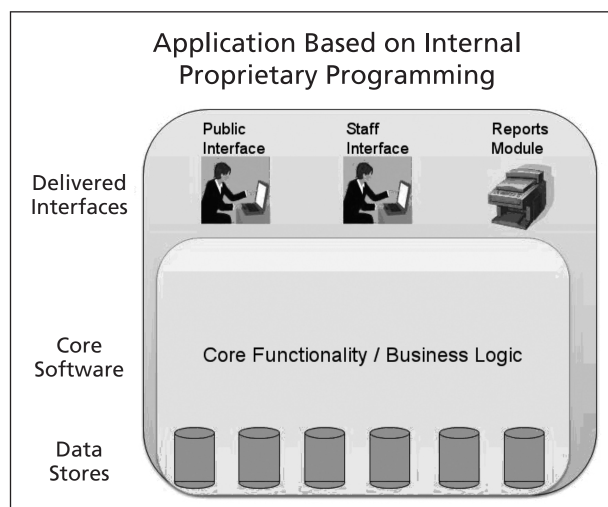

Figure 1
Proprietary programming without API.

The history of library automation is dominated by products that have evolved through many different cycles of software architectures; few systems built from scratch with new architectures of the prevailing age have emerged and survived. Equipping these evolved systems with APIs, especially in the form of Web services, represents one of the major factors in their forward progression. Those that fail to adapt to this expectation may find themselves less competitive as the library automation market moves forward.

One of the first steps forward out of the purely proprietary programming arena was for developers to take a more abstract approach to database access. Early automated systems managed data with proprietary methods. They stored, indexed, and retrieved data in the most expedient and efficient ways possible, usually involving database functions created by the developer of the application. Since almost all applications involve various aspects of data management, the idea of continually creating this layer of functionality within the application gave way to the use of third-party database management systems. The use of a third-party database management system allowed applications developers to focus more of their resources on creating higher-level functionality and less on reinventing lower-level data-access routines. The transition from database access routines written for specific applications to third-party RDBMS products often introduced a higher level of system overhead requiring more additional memory, disk storage, and processing power. These RDBMS products offered much more advanced functionality and scalability.

The use of a third-party RDBMS comes with cost implications. Products such as Oracle require a separate software license from the application software. When a library implements an ILS that uses a commercial RDBMS product, it must either purchase the level of license required or take advantage of existing site license that its organization may have acquired. The ILS vendor may also bundle the RDBMS license into its cost package.

The use of proprietary databases continues to some extent in library automation systems. Two of the major ILS products, Millennium from Innovative Interfaces and Symphony from SirsiDynix, were initially developed using proprietary databases. In both cases, the company offers its system with an option to use Oracle or another major RDBMS. For libraries that do not require Oracle for other reasons, many sites choose to use the proprietary database. This option avoids additional licensing fees

associated with Oracle. The proprietary databases can often deliver faster performance since they avoid much of the computing overhead associated with high-end database platforms like Oracle.

The emergence of these third-party database management systems gave organizations a much more powerful way to deal with information across many different applications. An organization could assemble a set of applications for each of its business needs, each based on the same underlying database management system. This arrangement would allow the organization to perform report generation, data mining, and other tasks that span multiple software applications, even applications created by different vendors. It also would allow the organization to hire a single expert, or database administrator (DBA), to manage data across many applications. A DBA will have very specialized training in the database management platform used throughout the organization and will optimize database performance, develop data integrity and disaster recovery procedures, and provide support to other programmers and system administrators in using, extracting, and manipulating data across all the applications.

Figure 2 illustrates an application where database access has been abstracted from its core business logic. The separation of the lower-level RDBMS though an API provides multiple advantages. To the extent that the application itself consistently operates through the API and avoids proprietary database-access methods, it becomes possible to support different RDBMS platforms. The database and the application can use standard connectivity protocols such as ODBC (open database connectivity) or JDBC (Java database connectivity) that allow programming code to be written independent of specific database products. Once one of these standard connectivity layers is in place, programmers can access the underlying data using SQL (structured query language), widely supported across all major database platforms.

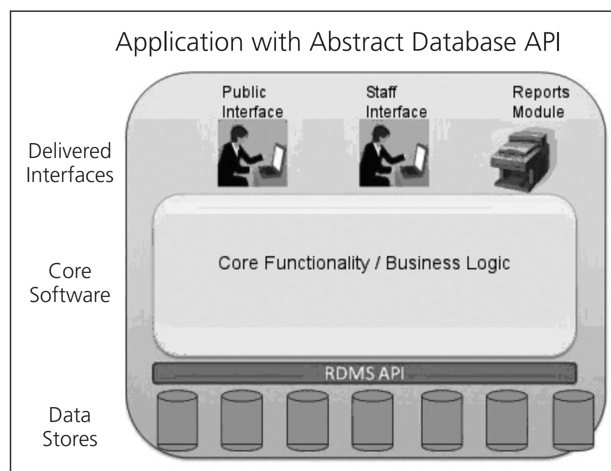

Figure 2
Application with RDBMS API.

Database programming involves tradeoffs between maintaining database independence and performance. Each database offers its own APIs to take advantage of features that boost performance and extend functionality. But when an application programmer codes to these vendor-specific extensions, the code becomes tied to a single database product.

As with other product categories in the tech sector, the RDBMS arena has seen massive consolidation. Many database products, such as Ingress, Sybase, Pick, Interbase, dBase, and others, have fallen away, leaving Oracle and Microsoft SQL Server as the primary database platforms used by library software. IBM's DB2 continues to see much use in the larger business arena, but has not been a major player in library software.

In the open source ILS arena, MySQL and PostgreSQL find wide use. We should note that Sun Microsystems purchased the open source MySQL product in January 2008, and database giant Oracle acquired Sun in April 2009, leaving the most widely deployed open source database in the hands of a company primarily involved with proprietary software.

The abstraction of the database layer from the applications layer of the ILS provides a great opportunity for a library to gain access to data in ways not limited by the software provided by the vendor. As long as the software vendor provides the customer library access to the schema of the databases involved and minimal documentation, the library should be able to, at a minimum, extract data and create reports in any way needed. As shown in Figure 3, it is possible to expose the database API outside of the application itself. It's also possible to modify and add records in a database though this method. These tasks require much more care, expertise, and knowledge of the software. An ILS is a very complex business system, with many interconnections among data elements. If the library modifies data in ways that interfere with any of the higher-level software, major problems can ensue. Software vendors may restrict these database operations outside their own software that change the databases.

It's also possible to have an abstract API with a proprietary database. But in practical terms, APIs tend to be used as tools for customer access to data primarily with the major industry-standard database products.

A variety of products are available to help programmers and librarians make use of data held in a RDBMS. Crystal Reports, for example, allows an organization to produce custom reports with many analytical features across any of their business applications. This type of reporting tool provides sophisticated access to data sets for those with a lower threshold of programming expertise. For organizations with experienced database programmers, one can write custom software to extract or analyze data.

While this approach provides the capability to access the underlying data through the APIs or other data access

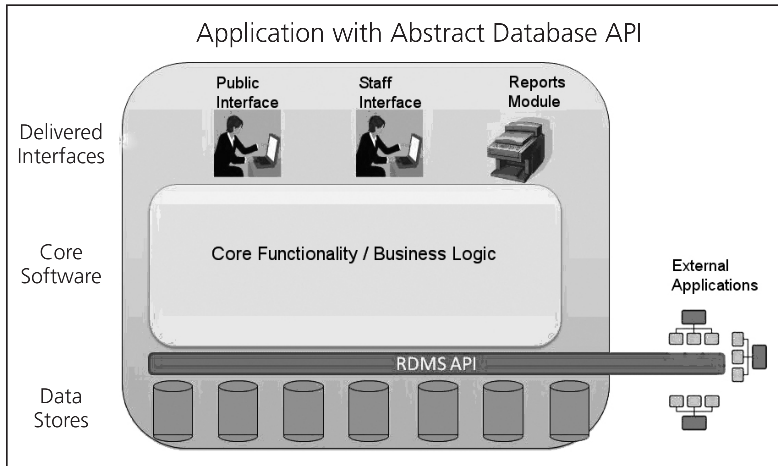

Figure 3
Customer access to database API.

routines associated with a RDBMS, it has limitations. It provides access to the raw data, but does not interact with the higher-level functionality of the automation product.

Offering APIs associated with the higher-level functionality of the ILS itself provides much more powerful capabilities. We're familiar with all of the features of an ILS offered through the staff and public interfaces. An ILS embodies hundreds, if not thousands, of business processes related to the operation of a library. The business logic of an ILS includes transactions for charging, renewing, or discharging materials; calculating circulation periods; loading records of all types; performing validation routines; purchasing materials; and performing search and presentation routines, to name just a few.

As software applications have evolved from monolithic proprietary designs to structured layered architectures, it has become common to use application programming interfaces internally. Figure 4 illustrates an ILS where the staff and public user interfaces as well as the reporting module do not access the business logic of the application through proprietary programming but only through the services made available through an application programming interface. This tiered approach has come into favor in recent years and is consistent with the service-oriented architecture. As applied internally within the application, having a presentation layer interfaces operate through a set of APIs to gain access to the underlying functionality and business logic provides many advantages, including the ability to rework user interfaces without having to reprogram the entire application.

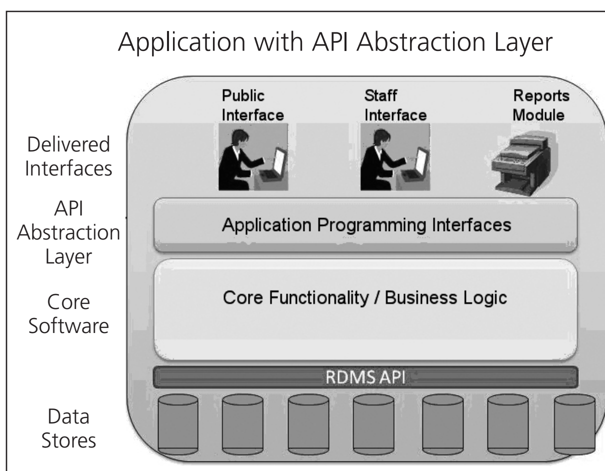

Figure 4
ILS with internal API.

Although an internal layering of software reflects good software design, it does not offer users of the software a direct benefit unless the APIs are made available to external applications in addition to their support for the vendor-supplied interfaces. Figure 5 illustrates a hypothetical system where the ILS follows a nicely layered design, with this important addition: the APIs that provide access to all of the data and services of the application have been published for use by the library. These published APIs might be accessed through scripts created by library programmers, through other applications that reside on the library's network, or through authorized applications beyond the local network. We'll explore some of the specific tasks that might be enabled through these APIs below.

Figure 6 describes a hybrid model, which corresponds more closely to real-world ILS products. This approach includes proprietary programming used internally within the application, but also includes a set of APIs that expose functionality to external resources. Applications that evolved from a legacy of proprietary programming may continue to use that code internally. Still, a subset of functionality is made available through APIs.

What Is an API?

An application programming interface involves a set of commands to which a piece of software will respond in a predictable manner. APIs live in the realm of computer-to-

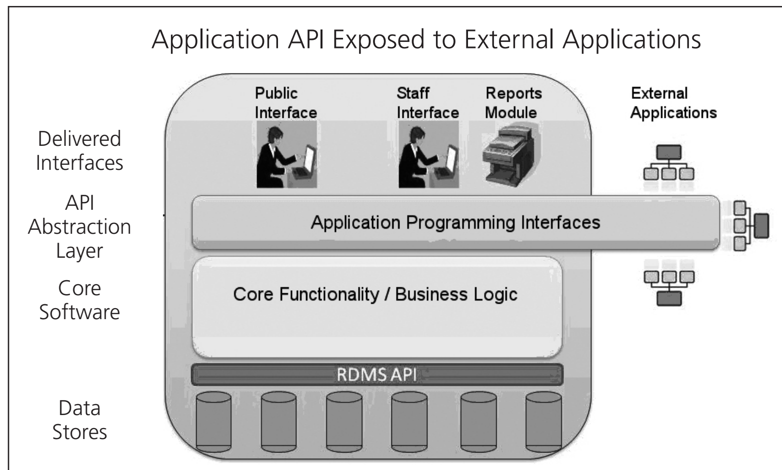

Figure 5
Exposed API services.

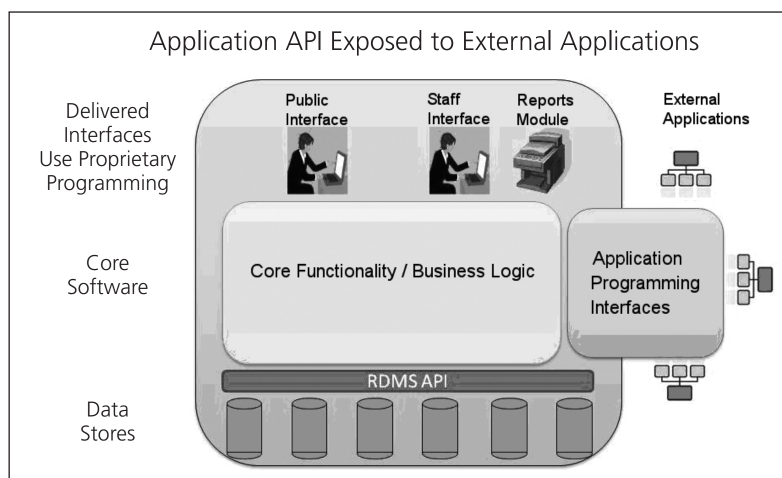

Figure 6
Mixed proprietary and API model.

computer interactions. The word *interface* in this context does not mean a user interface intended for humans, but rather mechanisms for one computer program to make a request from another.

APIs operate through requests and responses. The request might be a simple data lookup, or it might be a complex business transaction. In order for a programmer to know how to use the API, its creator must make available detailed documentation that describes each request supported, the syntax that must be followed, any mandatory or optional qualifiers, and the exact form of the expected response. Equipped with the full documentation of the API and its required transport mechanisms, a programmer will be able to write scripts, configure an external application, or perform other technical tasks that make use of the API.

It's also important to know what kind of communications protocols the API uses. Modern APIs will likely operate through a Web services model. Web services take advantage of the infrastructure developed for the Web to support computer-to-computer communications. Services and requests will be expressed in some flavor of XML and will use http as the networking protocol to transport the messages. Some environments use a more complex messaging system such as SOAP (simple object access protocol) to deliver requests and responses; others use the more simple REST (representational state transfer) which issues a request through a standard uniform resource identifier (URI). Continue to keep in mind that even when using Web technologies, these APIs operate behind the scenes. While one might be able to test a Web service implemented through REST, the actual use takes place between software programs.

Any application that supports an API will implement a responder that continually listens for requests, submits those requests to the underlying software components, and delivers the response.

Proprietary APIs

The concept of application programming interfaces has been around for many years, predating many of the modern conventions for implementing them today. In the ILS arena, we see examples of products that offer an API, but use a vendor-specific command language rather than the protocols and conventions more commonly used today, such as Web services delivered through REST or SOAP. These proprietary APIs provide many useful capabilities to extract data and extend functionality, but may be limited in their ability to support computer-to-computer interactions. Fortunately, the systems offering proprietary APIs (such as SirsiDynix Symphony) have also begun implementing APIs through modern transport protocols.

APIs and Open Source

Open source software has attracted great interest in the library automation sphere. At least two open source ILS products have become major contenders in the market, Evergreen and Koha. Open source software comes with the ability to view, modify, and redistribute its source code. This contrasts with proprietary software, where the

source code remains under the sole control of the organization that created it.

An open source ILS allows anyone to work directly with the source code of the application to fix any problems encountered, enhance existing functionality, or add new features. A key challenge to the maintenance of open source software involves governance issues that coordinate the work of a distributed group of programmers to ensure system coherence, consistent coding practices, and elimination of software bugs.

Open source ILS products benefit from customer-facing APIs as much as proprietary ones. Libraries using an open source system should not have to be constrained by the functionality of the delivered interfaces any more than those relying on proprietary systems. Nor should they have to become involved with advanced applications programming involved in the core of the application in order to gain access to data not addressed in the user interfaces.

Following an open source licensing model does not necessarily mean that an ILS is inherently more interoperable than a proprietary system. An open source ILS that does not offer support for standard industry protocols and a robust set of APIs will not be more inherently able to communicate with other library and nonlibrary infrastructure components than a proprietary system. Although an open source ILS might enable the library to develop the APIs needed for a given scenario that requires interactions with another application, such a development effort involves a much higher level of investment than being able to take advantage of existing service layers.

For these reasons, the same questions regarding the availability of APIs that apply to an open source ILS apply to the proprietary products. Like the proprietary versions, it may be the case that a given open source ILS product finds use primarily in the types of libraries that do not necessarily require this capability. As open source ILS products reach into larger libraries and more complex automation scenarios, their ability to offer this capability will become more critical.

Standards as Open Interfaces

National and international standards play a vital role in the way that libraries use their ILS. Most standards are implemented as a particular kind of API. Their presence in an ILS is a given; libraries must insist that any ILS they acquire adhere to the full complement of standards that apply to library automation. Any ILS should include support for standards such as

- MARC21: formats for bibliographic, holdings, and authority records
- Z39.50: search and retrieval
- SRU/SRW: search and retrieval of Web services
- OAI-PMH: open archives initiative protocol for metadata harvesting
- OpenURL: context-sensitive linking
- SIP2 and NCIP: protocols for circulation and patron data

As a point of clarification, in this report we look beyond the given standards as examples of API implementation. Library standards do not address many aspects of the data and functionality managed within an ILS. We're interested in APIs capable of accessing any aspect of the ILS.

API Security

It's important to prevent APIs from compromising the security of an application. The same level of control must be enforced when performing tasks on the system through an API as would be expected for an interface operated by a human. An expected part of interacting with an API will include authentication and authorization mechanisms. Any access to the parts of the system involving personal or financial data must be controlled through appropriate security, including the use of encryption before it is exposed to the network.

There may be some API requests that might be made freely available without restrictions. Some portion of the work handled by an API involves information that can be made publicly available, particularly in the open source ILS environment. Most libraries want the information in their catalogs to be accessed by any interested party.

Another aspect of concern relates to regulating the volume of requests that might be presented to the API responders. In order to prevent the deterioration of performance for the system's primary users, it's important to have security routines that throttle incoming requests to protect the system from misbehaved scripts or intentional denial-of-service attacks.

Terms of Service

One of the most important characteristics of an API has nothing to do with technology, but relates to its legal and business conditions. Any use of an API must be done with explicit permission. The terms of service specify details like who may access the API, any costs associated with its use, what they can do with any data they obtain, and any limitations on the distribution of intellectual property associated with the API or its underlying software.

A library using a proprietary ILS may need permission from the ILS vendor to enable the API. Some ILS vendors require that the library pay a separate license fee to make use of the API, some require that libraries using the API undergo specific training, and some require both.

Vendors that assume a certain level of responsibility for the performance of the system and the integrity of its data may be reluctant to turn over access to an API that gives the library the ability to work with the system in ways that might lead to unintended consequences that require major intervention. Some vendors justify an additional license fee for access to the API since it is a feature used by only some customers that requires ongoing development and support.

Another area of concern for vendors of proprietary software involves revealing specific details to competitors. The documentation of the API, and even programs that make use of the API, may fall within the terms on nondisclosure expressed in a software license. It's common for libraries within the community of users of a given product to collaborate and share scripts that make use of an API but to be restricted from sharing those scripts more broadly.

Programming done with the API of a proprietary ILS may also fall under terms of nondisclosure. While a company may be willing to provide its existing customers full information regarding the internals of its system, it may not want that information to be made available to noncustomers or competitors.

Open source ILS products do not have restrictive terms on any APIs associated with the system. Since the software itself is available without license fees and the source code is readily available, the issues relating to license fees and redistribution of intellectual property do not apply.

The second level of permission deals with how the library controls access to its system through the API. It might, for example, open up some API requests without restriction. Catalog search requests, availability of items, and other information that the library routinely makes freely available on the Web through its traditional user interfaces might fall into the kind of requests that would likewise be unrestricted through the API. Requests involving patron data or involving the financial records of the acquisitions module, however, would need to be controlled and limited to trusted business partners.

With Power Comes Responsibility and Cost

Adding a layer of APIs to a system that gives a library true access to the data and functionality of the ILS gives the library a great deal of power to work with its system. This power can be a mixed blessing. Once a library has the ability to read, and especially to write, into the data structures that underlie the ILS, extreme care has to be taken not to interfere with the normal operations of the system or to accidentally corrupt data.

The use of customer-created programming has implications for the support provided for the system by a vendor. If support agreements involve service-level agreements that guarantee high availability and expected time frames to respond and resolve problems, some caveats may have to be made when the customer has the ability to change data in the system in ways beyond the software delivered by the vendor.

The creation of an API introduces costs for its development, testing, and documentation and for support issues. It's common for vendors to offer training programs, custom programming, or consulting services related to customer use of these APIs. Like any other system component, once a vendor creates a set of APIs, the vendor must maintain it through each new release and attend to ongoing bugs and security issues. Since the creation and maintenance of APIs add cost and complexity to the system, vendors take different approaches to managing those costs. For some ILSs, especially those designed for large, complex libraries, the API may be considered expected functionality, and its cost may be folded into licensing or support fees. Other vendors take a more a la carte approach to API availability and support. Especially when only a minority of libraries make use of the API, a company may choose to charge a separate license and support fees for that feature.

Some ILS products may opt not to provide an API in order to offer low-cost products that can be more easily supported. APIs fall into a much lower priority level than affordable systems that can be used as delivered, particularly in the small to mid-sized library arena. For ILS products targeting this segment of the market, an API may not be an essential feature.

No Universal Expectation for APIs

The vast majority of libraries may have no expectation to work with their ILS through an API. Rather, they expect the system to provide for them a complete package of the functionality that they require to automate their work. Most libraries do not have the technical staff that would be needed to produce local extensions to the software using the API.

A reasonable degree of openness can be accomplished in other ways. An ILS with a full set of customization features should allow the library to adapt the software to its local preferences. All operational policies will be set; the presentation of online catalog should be malleable enough to adapt to the color schemes, logos, and other visual requirements held by the library.