Legacy Metamorphosis

By Charles Finley, Transformix Computer Corporation

Legacy Metamorphosis

By Charles Finley, Transformix Computer Corporation

Introduction

A legacy application is any application based on older technologies and hardware, such as mainframes, that continues to provide core services to an organization. Legacy applications can be stove-pipe monolithic, and difficult to modify. However, they can also be desktop information islands written in such products as Microsoft Access or Excel or many other desktop applications some of which have been neglected, abandoned or that have lost popularity. In the case of mainframe applications, scrapping or replacing them often means reengineering an organization's business processes as well. In the case of desktop applications, the costs to rewrite those that are still useful is sometimes not cost effective.

These legacy applications and desktop islands when taken together in an organizational context, point to a much larger problem for IT. It is called the stove-pipe effect. This is illustrated in Figure 1. Even though they are useful and, at least someone in the organization considers them worthy of retention, what makes them also a burden is that, in some ways at least they do not meet organizational needs. Moreover, these islands of automation can be expensive to own and operate. The integration needs and the variety of skills needed to maintain this monolith are strangling IT organization budgets. Here is a summary of the key areas in which legacy applications can be deficient:

- Development Tools and Rapid Development
- Stove Pipe Monolithic Applications
- Desktop Information Islands
- Web and Mobile Access
- Cloud
- Application Integration
- Database Sophistication
- User Interface
- Security, Integrity, Restart, Recovery

Even if from a functional point of view these applications provide value to the organization, there can be missing features (a gap) that would more closely align the

applications with organizational needs. The topic is explored further in the section Legacy Application Problems Addressed by Legacy Metamorphosis.

Legacy metamorphosis is about retaining and extending the value of the legacy investment through migration to a new platform. It also goes further in that it provides a standard set of tools with which to consolidate legacy and desktop applications on to a standard database management system with a standard and uniform set of development tools.

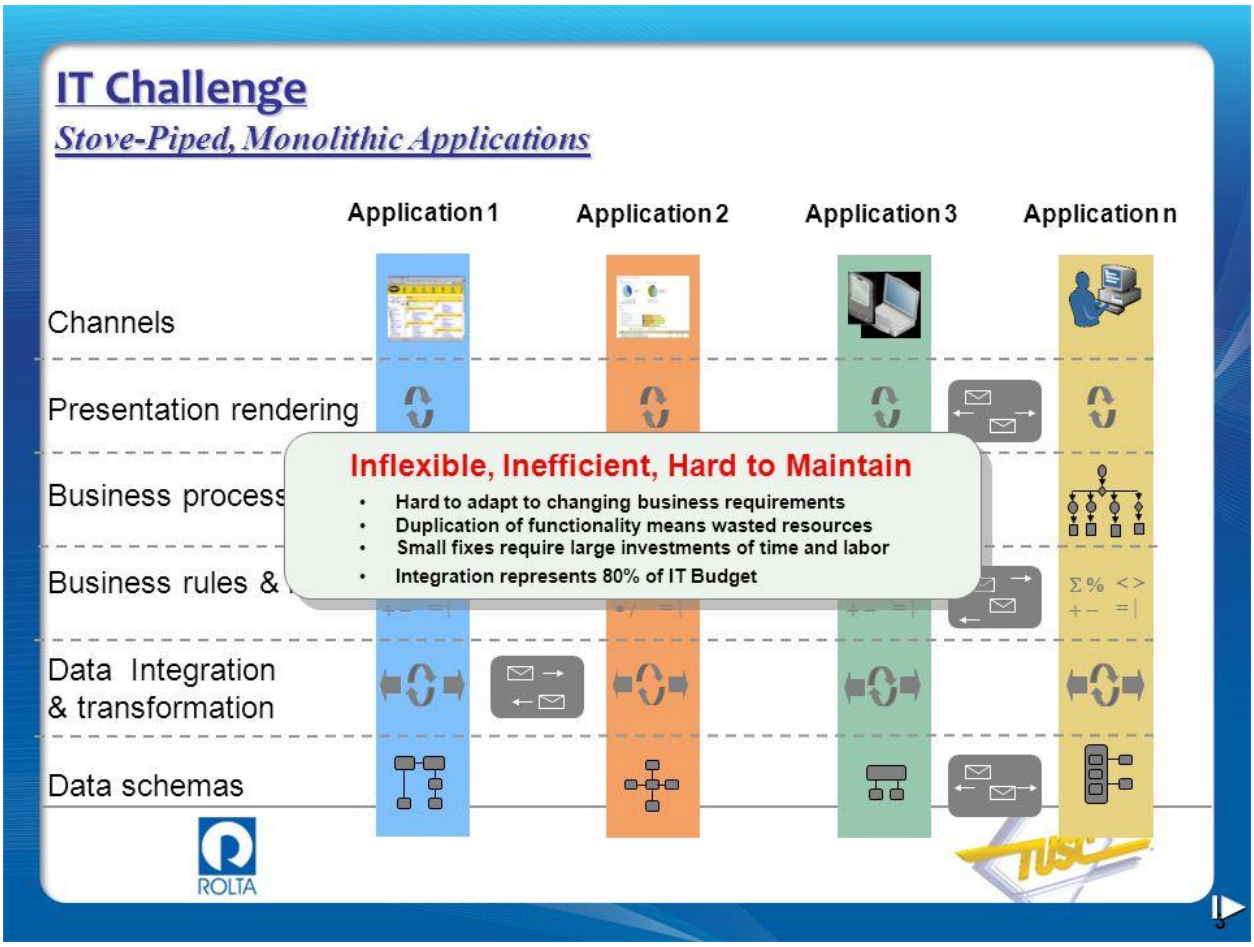

Figure 1 - Application Stove-Pipes

Migration versus Modernization

The terms application or database migration are sometimes used interchangeably. Unfortunately, there is no standard definition. At Transformix and for the purposes of this paper we use them in the following ways.

Migration

Legacy application migration is the process of moving an application program from one environment to another with minimal changes to the application. Similarly, legacy database migration is changing database management systems. The key is that program behavior and logic flow do not change much. Therefore, in a migration the original look and feel are retained.

Modernization

Legacy modernization, or software modernization, refers to the conversion, rewriting or porting of a legacy system to a modern computer programming language, software libraries, protocols, or hardware platform. This involves more substantial changes. To the application. Database modernization allows for the addition of features not used in the original application design. For example, indexes can be added that were not in the application before or views or stored procedures can be added.

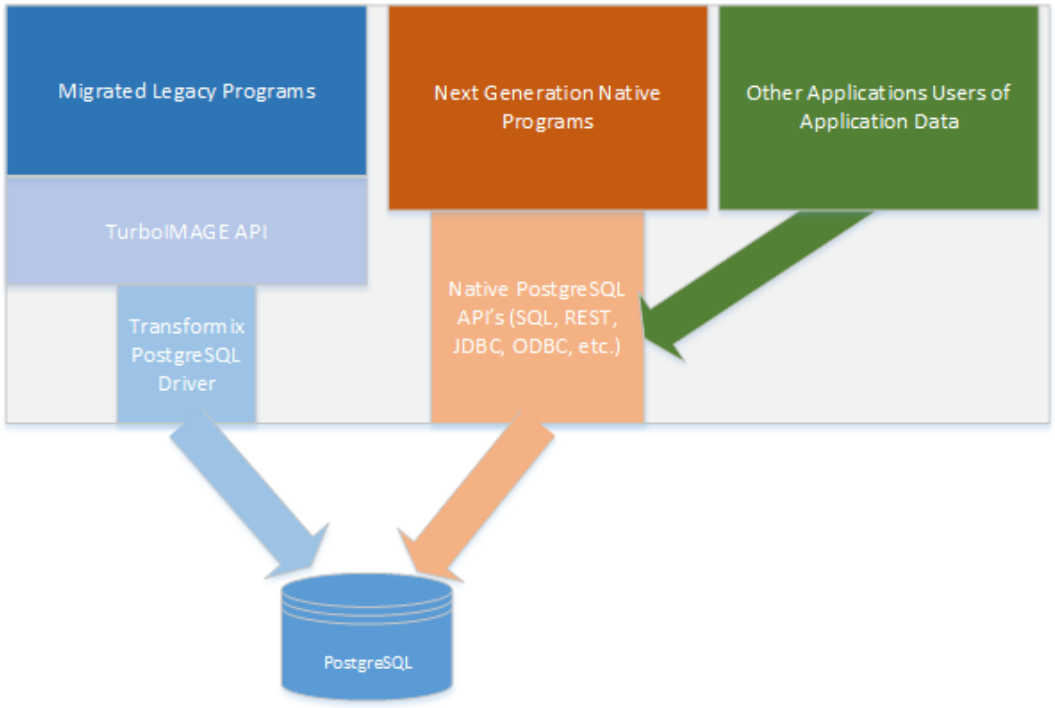

Figure 2 - Database Only Migration or Application and Database Migration

Application Metamorphosis

There are four possible steps in this approach which makes it appear similar to a metamorphosis:

1. Database Only Migration
2. Application and Database Migration
3. Database Modernization
4. Application and Database Modernization

It is important to note that throughout each of these stages from an application user's point of view, the application can still be used as it was originally designed and all of the original programs still work.

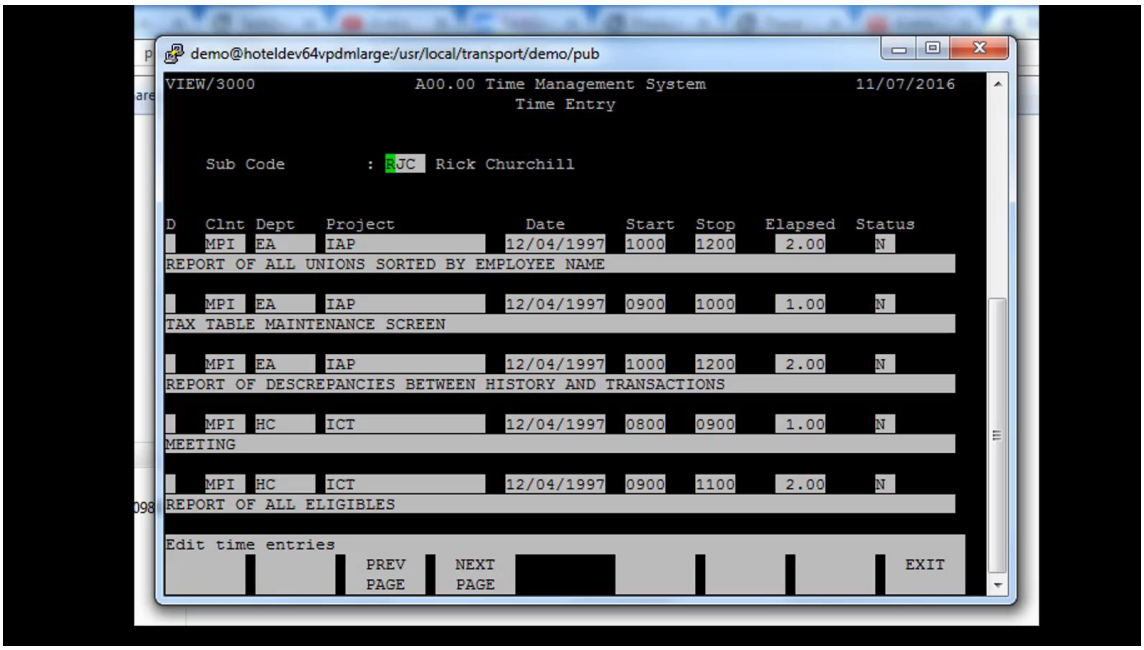

Figure 3 – Application and Database Migrated Application

Database Only Migration

Legacy metamorphosis is a database-first and database-centric approach. Figure 2 illustrates the approach. A key to the legacy metamorphosis approach is the relational database. In this approach, the application artifacts (code, JCL, data, etc.) are migrated to the target platform in a like-for-like fashion. By that we mean, the application users see the application as it was before with the same look and feel initially and program behavior. The legacy applications continue to work as they once did through use of a driver and legacy database emulation layer that implements the legacy database API calls. It is important to note that as soon as the application database is migrated in this manner, any number of commercially available off the shelf (COTS) applications can get access to the data. Further, since the data is stored in an RDBMS, it inherits the features available with any other applications that are RDBMS based. This means that both application specific next generation application code and other

outside software and applications that are not in this same stove-pipe can use the shared data source. Therefore, the use of the database frees the data from the stove-pipe, makes application integration simpler and allows the application to be extended in a non-invasive manner.

Application and Database Migration

Once a database has been migrated to an RDBMS, the application can be transformed in ways that allow it to perhaps change in platform (HP 3000 to Linux) or change programming languages. However, if for example the application already runs on Linux, the change could simply be a change in COBOL compiler. Figure 3 shows an example of an application from and HP 3000 application that has been migrated to Linux. As you can see, from the user's point of view, the application has not changed.

Database Modernization

Database Modernization are changes that are not required to make the original application work with legacy code. They are changes to the database that can be used to enhance its functionality. This involves the advanced use of features such as additional indexes (composite, for example), textual searching, views, stored procedures, functions, user defined functions, triggers and constraints, etc. The main motive for these changes is to allow newer programs to more easily become data centric instead or program centric in their architecture. This means that most legacy programs contain a lot more code than database centric programs. In a program that relies on record level access to data in a file system such as ISAM, type checking and referential integrity are enforced in each individual program. In a database-centric application, the database does much of this. Modernization of the database adds features that facility database centric programming.

Application and Database Modernization

Finally, Application and Database Modernization is accomplished using a JAVA framework called Axelor. This allows the application to be extended in ways that make it unrecognizable from its original form. Figure 4 shows that Axelor can be used both to work with the original application to maintain and enhance it and it can also be used for new development. In this context, Axelor supplemented by software tools provided by Transformix becomes an overarching development umbrella that allows the existing original application to be incorporated into the more modern Axelor framework. Figure 5 shows the Axelor menu with the TimeEntry application included in its menu system.

Axelor

Axelor is a French based company that supplies open source modern and robust applications under the AGPL license. Axelor applications are:

- Cloud Ready

Thanks to their robust and scalable architecture, Axelor solutions are particularly suited for the Cloud.

- 100% Ajax Web

Axelor focuses on ergonomy, for a fast and easy handling / the richness of a heavy client, with the lightness of a web client!

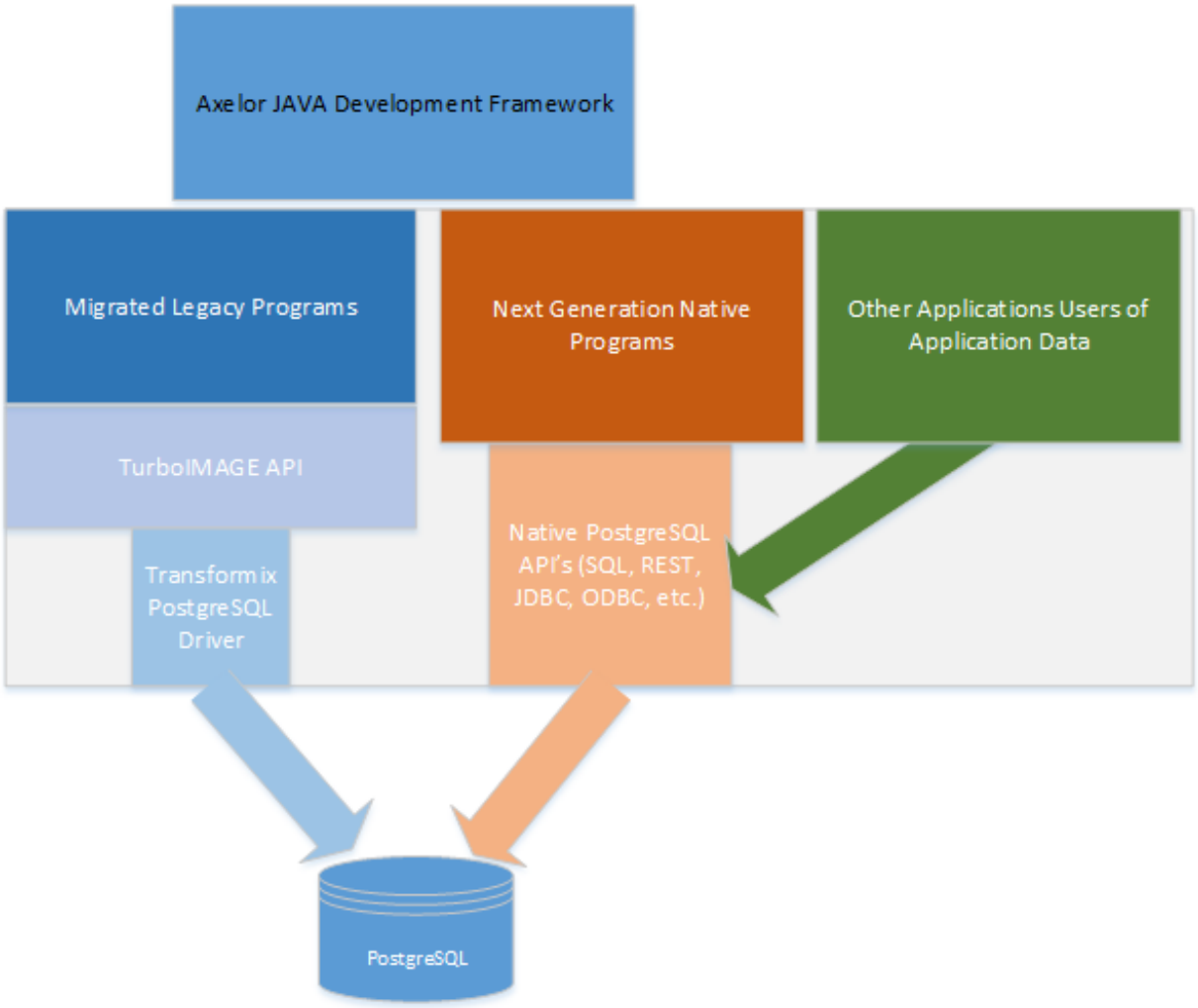

Figure 4 - Legacy Metamorphosis Model

Modern Architecture

Their business solutions are all built on a modular platform based on the latest JAVA technologies.

- Mobile

Axelor Business Suite has been designed to be fully responsive, in addition to its native mobile applications.

- An architecture based on Open Source Standards

An overview of their primary focus is provide here in a YouTube https://www.youtube.com/watch?v=6MShqGydkp8 video.

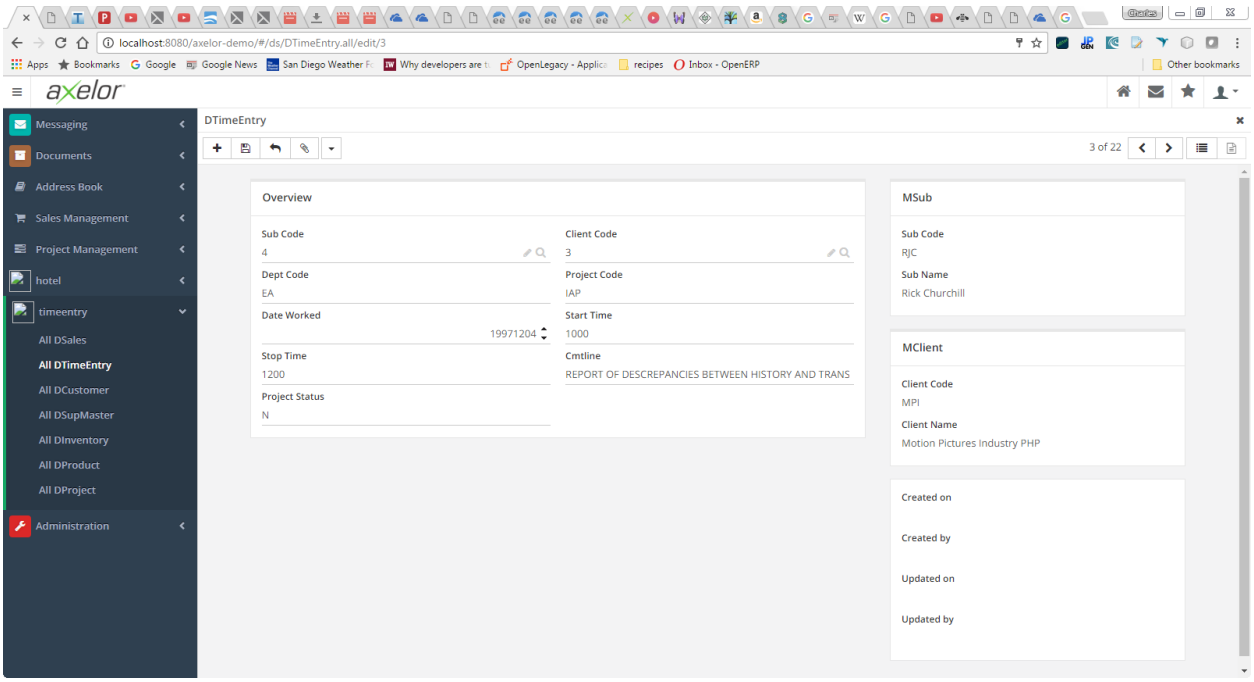

Figure 5 - Axelor with Original and New Application Modules

Axelor Development Kit

Axelor Development Kit is an open source Java framework for business application development. The Axelor platform is a rapid development framework for building business applications based on Java technology. It is easy to learn, customizable and saves critical time for complex applications. The application code is object-oriented, allowing you to use the standard Java APIs. The Axelor platform uses an approach to

model-driven development, where the heart of your application are Java classes that model your business code. This means that you can stay productive while maintaining a high level of encapsulation.

The primary RDBMS used by Axelor is PostgreSQL.

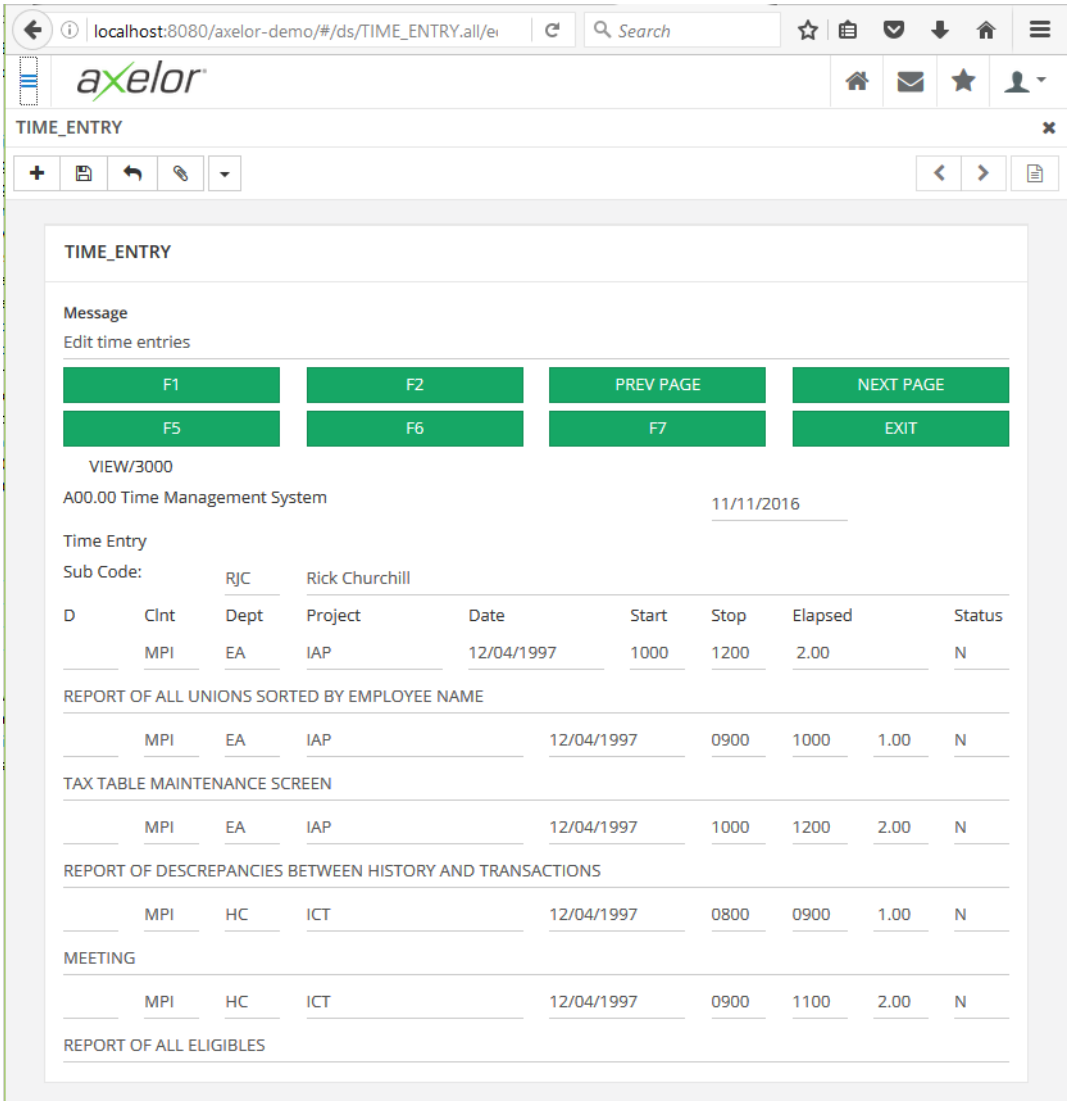

Figure 6 - Axelor Replacement for TimeEntry

Axelor Integration with Legacy Systems

The Axelor Development Kit is not designed to work with outside, existing databases. Transformix has adapted it for this purpose. Transformix provides five levels of integration with Axelor, they are:

1. Database
2. Forms
3. XMLRPC
4. Java Services
5. REST

These interfaces allow the legacy system to be integrated with Axelor and combined with the Axelor native applications and functionality.

Transformix also provides a forms editor that is compatible with Axelor. This is used to recreate legacy forms in an Axelor style.

Modernization in Small Steps and Metamorphosis

Legacy applications modernized with the Axelor Development Kit solution can then be selectively either replaced or supplemented with modern features. The screen shot in Figure 6 illustrates the point. This shows how the legacy application whose screen was shown in Figure 3 has been completely implemented with a modern adaptable look and feel. Both the original version and the new version work against the same database.

Once the Axelor solution is in place, immediate benefits start to accrue. Some of these are:

1. New application development can involve developers with older and newer skills. For example, a new module can be added to the application by developers who have never even heard of COBOL or the HP 3000.
2. Existing modules, screens, etc. can be replaced selectively. For example, an application that previously only worked on a green screen can now be made usable from a tablet or a mobile phone.
3. New functionality can be added to the application with features made available by modernizing the database. For example, full text search.
4. A separate related web server can get direct access to data either with database calls or a RESTful API.
5. The existing application can be slowly replaced in them with the users not even noticing.
6. Applications that were previously separate can be combined. For example, a legacy application and a desktop application that uses its data. The desktop application can become a separate module.

Legacy Application Problems Addressed by Legacy Metamorphosis

IT organizations accumulate applications. As these applications accumulate some are relegated to legacy status and many companies offer solutions to migrate or modernize these applications. Some of these solutions only focus on the user interface and some focus on adding web services, some require translation of all sources to some new language or framework. Like the applications they purport to replace, these solutions are stove-pipe solutions that only address one or two specific aspects of the issues IT management deals with. We take a step back and try to look at more than just the one or two issues. Our list to date is in this section.

Impact of Older Application Development Tools
Problem -- In-house application development coerced to fit capabilities and limitations of development tools, instead of meeting the needs of user organizations.
Stove Pipe Monolithic Applications
Problem -- Monolithic, ad-hoc application development creates artificial application boundaries that ignore enterprise-wide needs, and instead reflect budgeting process, sponsoring organization, and development organization assignments.
Desktop Information Islands
Problem -- Desktop documents and personal databases create ad-hoc, isolated, redundant, and conflicting islands of inconsistent information. Personal productivity applications using the familiar desktop metaphor (window on a document) are productive at a personal level, but very inappropriate to capture, maintain, or report data used by more than a single individual. Instead, business activities require shared use of widely available accurate, timely data available from single authoritative sources.
Web and Mobile Access
Problem – Either not available or difficult to implement
Cloud Hosting
Problem – Only selectivity available
Cloud Application Integration
Problem -- Either not available or difficult to implement
Rapid Development
Problem – Application relies on older toolkits that are not necessarily built for rapid development
Indexing Limitations
Problem – Expensive if available at all.
Lost History

Problem -- Applications rarely capture temporal nature of data or business objects, offering no capture of revision history, audit, and approval.
Relationships in Data
Problem -- Some important relationships hard coded (e.g. navigation buttons and links), other not pre-defined relationships often not available, or available only with user or programmer formed queries.
User Interface Training Burden
Problem -- Ad hoc, application specific user interfaces require user training and user application specialization.
Extended development cycles
Problem -- Extended development cycles, with applications upgraded in large disruptive releases.
Batch Application Integration Approaches
Problem -- Exports and batch feeds to interface individual applications.
Lack of Easy Application Integration
Problem -- New functionality does not integrate with other applications.
Large Variety of Technologies and Skills Required
Problem -- Application developers need to master too many technologies and specialties.
Problem -- Application developers lack full insights and mastery of business
User Interface Variability
Problem -- Conflicting application operation and user interfaces.
Data not subject to on-going review
Problem -- Data not subject to on-going review and update.
Multiple, inconsistent data sources
Problem -- Multiple, inconsistent data sources with no focused data authority and responsibility.
Difficult to interface with legacy applications.
Problem -- Difficult to interface with legacy applications.
Uncoordinated and nonintegrated Hosts
Problem -- Uncoordinated and nonintegrated departmental application islands vs. unresponsive centralized monolithic systems.

Legacy Metamorphosis Implications

Re-implementing applications on new platforms in this way can reduce operational costs, and the additional capabilities of new technologies can provide access to valuable functions such as Web Services and Integrated Development Environments. Once legacy metamorphosis is complete the applications can be aligned more closely to current and future business needs through the addition of new functionality to the transformed application.

In short, the legacy metamorphosis process can be a cost-effective and accurate way to preserve legacy investments and thereby avoid the costs and business impact of migration to entirely new software.

The goal of legacy metamorphosis is to retain the value of the legacy asset on the new platform. In practice this metamorphosis can take several forms. For example, it might involve translation of the source code, or some level of re-use of existing code plus a Web-to-host capability to provide the customer access required by the business. If a rewrite is necessary, then the existing business rules can be extracted to form part of the statement of requirements for a rewrite.