QORE TECHNOLOGIES

A TRULY AGILE INTEGRATION AND API MANAGEMENT PLATFORM

by David Nichols

An integration platform is a must-have for any company with digital transformation and business process automation aspirations, however despite the plethora of solutions available, a great number of such projects end up over budget, provide very little business flexibility for future changes, fail to reach performance, transparency, and reliability goals and suffer from a high TCO.

The purpose of this paper is to describe an alternative ap-proach for an agile platform for the rapid development of fault-tolerant, disposable interfaces and APIs, specifically aimed at providing maximum business flexibility and interface quality while reducing implementation and long-term operational costs, allowing companies to execute a digital transformation strategy while improving business flexibility and keeping costs firmly under control.

Software development approaches based on fixed up-front requirements and heavy change-management processes do not provide much business flexibility or customer responsiveness.

Agile methodologies aim to remove bureaucratic barriers to producing tangible results without sacrificing quality.

Effective API management requires maximum flexibility and a managed life cycle of deployed code.

Traditional development approaches (particularly in fixed price, outsourced projects) are normally based on a contractually-fixed set of requirements and relatively heavy change-control processes. Movements in the IT industry to formulate new development approaches to match more realistic business scenarios where requirements are continually refined and updated in the course of a project, have resulted in agile development methodologies, of which one of the most prominent Scrum.

Agile development methodologies provide a disciplined approach to IT development intended to produce usable results faster and with higher quality than traditional approaches. Fundamentally, agile methodologies try to remove bureaucratic barriers to producing tangible results without sacrificing quality.

Fast and flexible development of disposable interfaces and APIs is particularly relevant in modern integration architectures; services and APIs must reflect the current needs of the business processes they are servicing; changing business requirements often require corresponding changes to technical processes.

The majority of open-source and proprietary integration platforms are based on Java. Java provides a very powerful and complex platform for developing business solutions. However, Java's complexity can also be a drawback, as increased complexity trans-lates directly into increased risk and costs.

QORE TECHNOLOGIES

Manifesto of Agile Software Development

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

The framework must provide certain business-critical features "out of the box" to ensure these features are always delivered with every interface.

A radical approach to agile enterprise integration: focus the expressive power of a domain-specific program-ming language within a rigid frame-work that guarantees reliability and transparency of interfaces and services.

When increased complexity is countered by increasing the size of the development teams, larger project teams require exponentially greater coordination effort as the number of developers grows. Eventually, costs get out of control, and the flexibility to react to changing business requirements is lost.

Furthermore, many software projects are planned with a life span of three to five years in mind to amortize the cost of development. This is not realistic for most business process integration projects. More flexibility with a smaller investment is needed.

Instead of using a language and application framework designed to enable large projects teams to collaborate, the language and application framework should empower smaller teams to do more in less time. To better achieve agile development goals, the language and the application framework should do everything possible to remove bureaucratic obstacles to producing tangible results and should instead facilitate rapid prototyping and the generation and controlled decommissioning of disposable interfaces and APIs.

As Ruby on Rails provides an agile development frame-work for web applications, this paper aims to describe a similar framework for the development of integration and business process automation projects. While Ruby on Rails leverages the expressive power of Ruby, an object-oriented, dynamically-typed interpreted language, within a web application framework, allowing for very fast implementation of powerful web applications and openended customization of the behavior of all elements of the system, **this paper proposes the integration of a domain-specific interpreted language within a rigid framework to facili-tate the rapid creation of fault-tolerant interfaces and APIs, in short, a truly agile integration platform for business process automation and digital transformation.**

This framework must do more than satisfying basic technical needs such as database connectivity, transactional integrity, data transformations, protocol conversions, and messaging integration. These subjects and more must be covered by any integration platform. In addition to this, the framework must provide maximum flexibility to changing requirements, robust error-handling to ensure reliability of the solutions delivered on the platform, operational and configuration transparency to enable quick identification of the flow of logic in interfaces and errors in the source code, and must meet performance needs without requiring exorbitant amounts of hardware to do so.

Enterprise integration platforms should be **flexible**, **reliable**, **transparent**, and **performant** to provide true value to business customers.

If these features are not guaranteed by the framework, then in typical IT projects when budget and/or time are tight due to unclear or delayed business or technical requirements, underestimating the effort required, or other problems common in real business environments, these critical features are often postponed to a post-launch phase, and, in the worst cases, never delivered at all.

Furthermore, these features are even more critical in an agile development process, because, as long as they are satisfied by the framework, they do not take the form of bureaucratic obstacles to generating tangible results and can never be postponed to a later delivery phase. The presence of these features in the framework keeps programmers focused on generating only the code that must be customized for the particular job at hand.

From the point of view of business customers, the framework must provide the following high-level features:

- **Flexibility:** quick response to changing business requirements: short development times while maintaining high qualit
- **Reliability:** unavoidable technical errors such as application, da-tabase, and network outages are handled gracefully and are re-covered automatically when possible; the possibility for error recoverability must be guaranteed by the platform by storing execution status and providing a mechanism for correcting errors and automatically recovering the suspended tasks

BPA Solutions should be flexible, reliable, transparent, and performant to provide true value to business customers.

- **Transparency:** the status, configuration, and even the source code of automation tasks is readily visible and easily accessed through an operational UI and network APIs to make changes easier and to facilitate a short feedback loop for change control.
- **Performance and Scalability:** must not require exorbitant amounts of hardware to meet performance expectations; solution should scale on today's and tomorrow's multi-core and multi-processor virtual/cloud and physical hardware.
- **Security:** must support enterprise authentication and authoriza-tion with fine-grained permissions so that APIs and data can be protected appropriately; this is particularly important for data protection legislation such as the EU's GDPR.

These features can be realized by executing user-defined code written in a domain-specific programming language within a

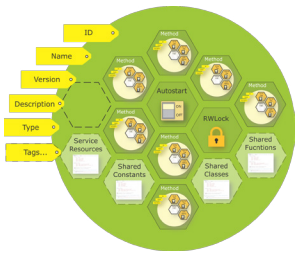
rigid platform that provides fault-tolerance, transparency, and standardized error handling and operational reporting. Such a system would make a powerful and flexible platform for agile de-velopment in enterprise environments with high process quality and a low TCO.

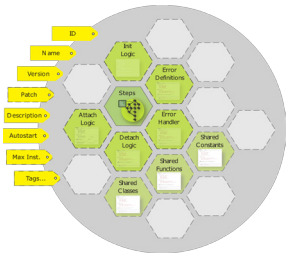
In such a system, both integration tasks and services would be made up of objects with code attributes written in this language, executed in carefully controlled and limited contexts the integration framework that provides fault-tolerance and transparency.

The result of embedding a domain-specific language in a rigid integra-tion and API framework: a powerful platform for agile integration enter-prise environments with high process quality and a low TCP.

In this way, the inherent flexibility and expressive power of the domain-specific language are embedded in a framework that covers of all the bureaucratic elements of interfacing such as: dy-namic loading and unloading of APIs, tracking statuses, data and external connections, facilitating synchronous and asynchronous messaging, providing shared data and messaging resources to the system, and more. This results in a system that provides the quick reaction and development times to stay on top of changing business and technical requirements while providing maximum reliability and fault-tolerance of the automation tasks built upon it.

The structure of the platform, when coupled with executing code in the domain-specific language in limited and controlled contexts as attributes of integration objects, will tame the expressive power of the language and focus it narrowly on the integration task at hand, allowing rapid development of powerful fault-tolerant “reliable-by-design” interfaces and equally powerful disposable APIs.





A graphical depiction of disposable workflow and service objects with embedded logic attributes written in a dynamically-typed language.

Therefore, the best of both worlds could be achieved: rapid development times leading to increased business flexibility and reduced development costs, while the inherent reliability of the solution leads to increased process quality and customer satisfaction, also while reducing long-term operational expenses.

To achieve this goal, the domain-specific language would have to be designed to support a unique set of features. Furthermore, the platform would have to be tightly integrated with the language, and would have to have the correct design to deliver on promises of true automated error handling in order to provide the high levels of reliability to allow large volumes of complex tasks to be managed by small operational teams.

The high-level, domain-specific language would have to have the following features:

QORE TECHNOLOGIES

The platform would manage the lifecycle of disposable APIs and inter-faces.

The domain-specific language must support at least the unique feature set described on the right to enable safe execution in a structured environment, focused on integration tasks.

This platform would be an intermedi-ate step between a scripted applica-tion and an open-ended agile frame-work such as Rails or Zope.

The language so designed would permit a robust operational framework to be developed featuring automatic error handling and live upgrades of interface and API implementations.

- **Powerful Integration Capabilities:** Database access with trans-action management, character encoding, XML and common protocol and web-service protocol support, publish-subscribe and point-to-point messaging capabilities, and more.
- **Logic Embedding and Sandboxing Support:** Visibility of data in the embedded code must be controllable; the platform must restrict access to features of the language that could compromise system integrity. Embedded code must be containable in discrete objects that can be created and destroyed on demand, and code objects must not interfere with other objects except by design in a controlled and safe manner.
- **Resource Control/Tracking:** Allows the framework to catch and log programming errors and automatically free resources; to provide a safe execution platform for the domain-specific code.
- **Clean Threading Model, SMP Scalability:** The solution will require very fine-grained threading to be scalable on current and future multi-core and multi-processor cloud, VM, and physical systems.
- **Exception Handling:** Allows high-level, complex actions to support robust automatic and user-defined error handling.

This feature set should allow an integration platform to be developed that provides a safe framework for the execution of code written in a domain-specific language designed to facilitate enterprise integration by focusing the expressive power of the language solely on the specific logic required to customize inte-gration tasks.

The platform must track the execution of each element in an interface's flow as executed from dynamic user-defined code and react when errors occur to enable error conditions to be automatically recovered or, in exceptional cases, to be flagged for manual intervention. Additionally, the platform should pro-vide a flexible automated solution to complex error conditions, such as lost response messages to successfully-processed out-going messages with a non-repeatable action (for an example, see the callout on the left). Furthermore, the p should provide a structure for handling difficult tasks such as asynchronous messag-ing/event processing in safe way where the system itself performs most of the hard work and the programmers simply define code attributes in pre-defined objects.

To provide for maximum efficiency and to allow for the managed lifecycle of disposable interfaces and APIs, custom code must be dynamically loadable (from a database for example) and deleted on demand, meaning that the system must support online logic upgrades while guaranteeing continuity of serice (even while up-grading interfaces or APIs) as well as the consistency of the plat-form.

Such an implementation is less open-ended than an application server or even an agile platform like Rails. However the structure of the system serves to focus the code purely on the tasks that must be customized, and takes programmers further toward the goals of agile development than less open-ended systems in the sense that more overhead/bureaucratic work is removed from the programming domain and placed in the system domain. In this way, this solution could also be considered an in-termediate step between application scripting (for example, the Singl.eView billing platform, which uses its own interpreted language to customize the behavior of the system in controlled contexts) and open-ended agile platforms such as Rails.

The integration and API management platform described here is only useful with predefined interfacing elements that can be customized by plugging in code as attributes of the objects, and therefore is a step beyond application scripting, where the application is normally usable without user-code customizations, however, it gives more structure and focuses the user code more narrowly on integration tasks than Rails. Rails and Zope, another open-source application server based on Python, another interpreted object-oriented scripting language, must be more open-ended due to the nature of an application server and web application development in general.

Example of a complex error condition: a billing account for a new customer can only be created once; the account is successfully created, but the response message is lost due to a network problem; this type of error must be recoverable by the platform without stalling the workflow by requiring manual intervention.

However, the interfacing and service framework described here, coupled with a web-service-based API and a zero-install web-based client UI giving complete access to the status and configuration of the system, could potentially give businesses the power and flexibility to realize their automation projects with significantly reduced one-time and long-term costs and with much improved flexibility versus traditional solutions, even if agile development processes are not employed.

One concrete example of a system taking such an approach is the Qorus Integration Engine® by Qore Technologies. This innovative system embeds the Qore programming language, which is an open-source, object-oriented, interpreted, dynamically

QORE TECHNOLOGIES

lytyped domain-specific programming language that was specifically designed for the realization of an optimized, fault-tolerant integration platform for agile enterprise interfacing and API management as described in this paper.

Qorus allows for total encapsulation of Qore-language code as attributes of discrete objects (for example, the main logic in a step in a workflow); the code is loaded on demand from the system database schema and only executed in a controlled manner in the appropriate context and only given access to the relevant data being processed.

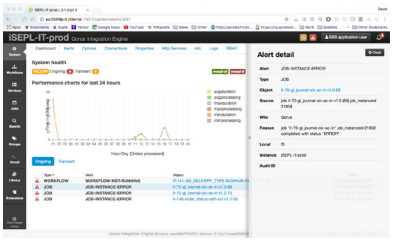
The lifecycle of all user objects in the system is managed in a way that permits live commissioning, upgrades, and decommissioning of interfaces and services. Furthermore, the server process is highly threaded and highly scalable on multicore and multi-processor systems, in a large part due to the unique internal architecture of the Qore programming language. Additionally, standard web-service protocols are used to export all system functionality to network clients making integration easy with standard technologies and in existing enterprise environments.

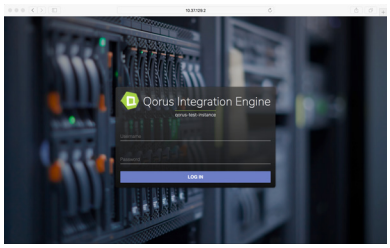
Complex multithreaded interface definition is very easy; the Qorus system takes care of all the hard parts of managing threads, statuses, data, asynchronous messaging/events, remote connections, data conversions, error recovery, and more, freeing programmers from the burden of managing these tasks, and focusing their efforts only on the code that must be customized to realize the interface or service being developed.

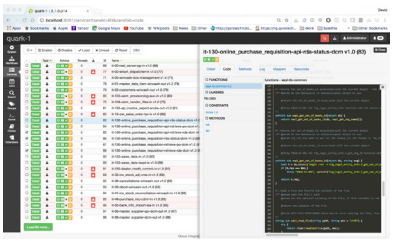
Qorus supports automatic error recovery even of com-plex conditions such as lost response messages to non-repeatable actions (as defined earlier in this paper) by defining the appropriate user-code attribute of a workflow step. Due to such features, Qorus supports automatic fault-tolerant execution of stateful interfaces, meaning that, as long as simple design criteria are met and network transports and end applications are eventually available, interfaces are guaranteed to be process their order data to completion.

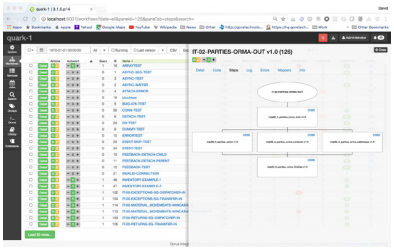
Qorus provides the rigid structure for providing rapidly-developed APIs and powerful, fault-tolerant interface execution, while embedding the Qore programming language in discrete objects as user-code attributes of API, interface, and data conversion

Qorus Integration Engine® was designed to facilitate the agile development of disposable interfaces and APIs following the approach described in this white paper.









Qorus' responsive zero-install event-driven web UI and web service APIs publish the status, configuration and even the source code of user objects to provide businesses with more information and to shorten the feedback loop when changes are necessary.

objects to support agile development in business process au-to-mation tasks by providing an unprecedented level of flexibility to react to changing business requirements.

This approach and Qorus Integration Engine® itself has proved to drastically lower development and operational costs while improving reliability and providing very high business flexibility in real business environments. For example, Hutchison Drei Austria successfully re-worked their entire Business Support Systems architecture, using Qorus as the main integration engine for all business-critical processes requiring IT automation. A small team of six developers with no prior integration experience was able to provide flexible fault-tolerant integration solutions for this major Austrian telecommunications carrier in less than six months from design to live production use.

The Hutchison group uses Qorus to provide global integra-tion services to hundreds of internal and external systems in a massive global integration project covering over 52 million end customers in Europe with high business flexibility and very low development and operating costs.

For more information on this approach or for information about the Qorus Integration Engine®, contact Qore Technolo-gies at info@qoretechnologies.com, +420 222 521 165, Prague, Czech Republic.