Enterprise Integration Modeling

A Practical Enterprise Data Integration and Synchronization Solution

Mihaela Iridon
Cândea LLC
Dallas, TX, USA
e-mail: iridon.mihaela@gmail.com

***Abstract*— As line-of-business software systems take shape and evolve over time within an organization, so does the need for such systems to interact with each other and exchange data, making it imperative to design flexible, scalable integration architectures and frameworks to support a robust and well-performing enterprise system. System integration is a multi-faceted undertaking, ranging from low-level data sharing (Shared Repository or File Sharing), to point-to-point communications (Remote Procedure Invocation via Service Orientation), to decoupled data exchange architectures (Messaging). It is common to build entire integration sub-systems responsible not only for exchanging information between systems (commands and notifications) but also for potentially more complex business logic orchestration across the entire enterprise (Message Broker). This paper is contemplating a practical data notification and synchronization integration solution that allows multiple enterprise domains to share data that is critical for business operations. The article presents a real-world integration architecture achieving this business objective, together with the corresponding system models and design artifacts, and shows how the data integration is realized using a broker-based messaging approach employing various enterprise integration patterns.**

Keywords-Enterprise integration; system modeling; data integration; canonical model; integration patterns.

I. INTRODUCTION

Within an enterprise, system integration solutions are almost always designed and implemented as an afterthought, as an attempt to build or to expand a new or existing enterprise architecture comprised of heterogeneous legacy system. It may be safe to say that most companies do not start off with an integrated enterprise architecture but rather a core domain (also referred to as a vertical), which will eventually grow and become part of a larger enterprise system. In many cases, such integration is achieved by employing various off-the-shelf integration products, such as Microsoft's BizTalk [7] or TIBCO.

Software system integration comes in different flavors, depending on the business objectives, the overall enterprise architecture, and ultimately the realization approach chosen. In Section II we will investigate these driving factors and then present a concrete implementation approach and its models in Section III, as it has been proposed and adopted by a provider of the nation's largest portfolio of benefit and payroll products and services designed to help more than 200,000 small and medium-sized businesses.

This paper presents a data integration and synchronization blueprint aimed at implementing the "Maintain Data Copies" data integration pattern [8] by means of a decoupled integration mechanism realized on a custom broker-based messaging architecture [10] [12]. The data payloads exchanged between the loosely coupled sub-systems abide to a *ubiquitous* integration language, referred to as the *canonical model* [7] as described in Section IV. This model is the unified abstraction of the data structures that must be shared and synchronized between these systems.

II. COMPARING AND CONTRASTING FUNCTIONAL AND DATA INTEGRATION

When building a large enterprise software system by bringing together multiple domain applications, the first question that must be answered involves the level of abstraction at which the integration specifications are being defined: Do the sub-systems only need the data that allows them to carry out their own functions, or do they also require access to cross-domain exposed functional features? In other words, should a system expose data only or features as well?

The answers to these questions will determine the type of integration that must be realized: data or functional integration, and, perhaps even further, it will help discern between the need of a flexible, lightweight, loosely-coupled integration architecture and one that adds enterprise features and interactions, transcending domain system boundaries. It is also possible that, in some cases, a hybrid approach may be pertinent, either to realize a quick and simple integration with a narrower scope (e.g. a test product implementation), or to overcome deep architectural and data model discrepancies between the existing systems. In this case, the solution must fulfill some imperative enterprise needs - whether they are related to exposing new system features in a short amount of time or at a lower cost until further market research proves the worthiness of additional funding for a comprehensive, scalable, extensible, and suitable solution.

A. Functional Integration

This type of integration involves exposing data and behavior [9] to systems that participate in the integration in order to trigger or invoke business features exposed by these systems. Usually, a pure Service Oriented Architecture (SOA) [3] [4] would be the simplest architectural approach

 ISBN: 978-1-61208-448-0

that could realize this requirement, but it would introduce system coupling and would not be easily scalable [5]. Web Services implement in effect the Remote Procedure Invocation integration pattern paradigm [7] and this implies mutual awareness of the presence of – and the functionality provided by - each of the integrating systems.

Complexity becomes apparent when more than two systems must interact at a logical and/or functional level of abstraction by invoking these exposed features and generating chattiness across the network, or when systems evolve, possibly threatening the stability of the integration contracts and hence of the solution. Several options are available to alleviate these problems, from architectural ones to following best practices and proper functional decomposition and service encapsulation, and eventually to making the proper technology choices [4].

B. Data Integration

This type of integration assumes that the various integrating systems were not designed to work together [1], and that they do not have direct access to the entire enterprise data but only to that which they provision directly. These systems were built in order to fulfill certain functional and business requirements, rather than architectural ones. It is also possible that some systems were acquired at a later time (e.g., corporate mergers, third-party software acquisitions, etc.)

Given that the systems evolved independently, enabling them to interoperate using multiple copies of the enterprise data (i.e., multiple data sources) while providing enterprise-level business features in a unified fashion is problematic, since there is no single source of truth and, potentially, no single source of data entry. Multiple applications may allow users to enter the same type of data from different user interfaces that sit atop of different business/logic layers and, consequently, different data sources.

Achieving this type of data integration can rely on either custom solutions (for example, involving an enterprise service bus), or commercial tools (such as implementations of a Master Data Management system), which may expedite the time-to-market of such an integration, sometimes at lower costs than custom solutions [2] [7]

III. A PRACTICAL DATA INTEGRATION AND SYNCHRONZATION SOLUTION

Consider three major business domains, Human Resources (HR), Payroll, and Benefits. The common ground for all three is the demographic data that defines the companies (or clients) that these systems are servicing and their employees. As is quite often the case, neither domain was built with a true enterprise vision in mind, neither architecturally, nor functionally. Yet the main enterprise data on employees and clients served must be shared across all domains when multiple data copies exist, one per domain. These data sources were designed for a very specific purpose, making it prohibitively expensive to refactor the systems' layers and the business applications so that they rely on a single source of truth – a unified data source across the enterprise. A solution employing Master Data Management (MDM) tools has been evaluated but the business requirements did not warrant such elaborate implementations for this particular case. The proposed and agreed upon solution was to implement the "Maintain data copies" data integration pattern [8] by means of a custom scalable and extensible middleware architecture (or integrating layer [10]), reusable frameworks and models, and carefully-chosen technologies, to fulfill the business need of providing multiple services (HR, payroll, and benefits) to an array of small to large size clients.

The following subsection presents the main models of the proposed integration solution, where data notifications are being exchanged between the various domains via a broker-based messaging architecture, using various enterprise integration patterns, as depicted in the EAI pattern mapping diagram in Figure 4. The data payload for these messages is wrapped inside a context-based notification model, allowing participating systems to take the appropriate action – based on their own domain rules – using the data received from the message broker. The individual domain systems are not aware of each other, only of the message broker through which they communicate.

A. The Integration Models

All models, structural and behavioral, included in this paper are excerpts from the technical design specifications document created on behalf of the client's Enterprise Integration Solution [12] and they are being used hereby with permission from this client.

1) Structural Models: High-Level Enterprise Integration Architecture and Components

The integration middleware was designed as an extensible, highly-responsive, and scalable broker-based topology through which the integrating domain systems will exchange data notifications in near real-time and in a loosely-coupled fashion. The middleware is built on durable messaging frameworks, such as an enterprise service bus (ESB), queues, an entity mapping/correlation infrastructure, and various service endpoints (SOA).

The high-level component diagram (Figure 1) shows the three business verticals as clients to the enterprise services that provide access to features that implement cross-cutting concerns (logging, SSO, audit) while indirectly exchanging data notification messages among each other, without awareness of each other or the features they provide, using the integration middleware exposed via a service endpoint (i.e., the Data Notification Receiver Service). This design ensures system scalability and plasticity of the integration scope (data or functional), while hiding the actual technology specifics from the systems that participate in the integration.

2) Object/Data Models: The Canonical Model

The data notifications exchanged between the systems via the service-broker integration middleware is a two-layered object model, with (a) the actual data payload represented by the integration ubiquitous model, also referred to as the Canonical Model [7], and (b) the notification model which is wrapping (or encapsulating) the canonical model payload, adding context, source, and target details to the communication messages.

 ISBN: 978-1-61208-448-0

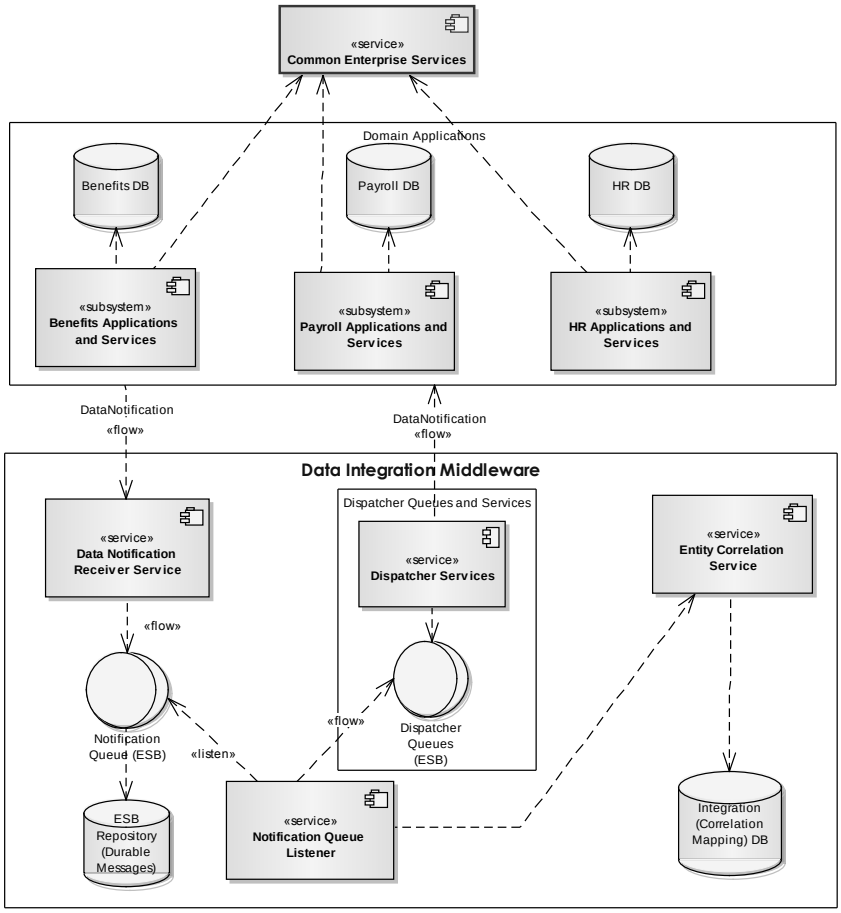

Figure 1. Overall enterprise integration topology: business verticals and integration middleware

This allows for a reusable notification model, where - by employing generic data types for the payload wrapped within the notification together with the appropriate inheritance (generic type inheriting from the non-generic type) – we can design any number of notification schemata that could encapsulate any business entity models inside a generic payload. The payload is domain-specific (or enterprise integration-specific in this case), whereas the notification model is domain-agnostic. This is depicted in the object model in Figure 2. The generic type T of the payload can be anything that one would define for a given domain: employee, client, address, benefit, participant, dependent, etc. In fact, a separate object model for the enterprise integration has been defined and is used in the implementation of this solution (see the Section IV for further details).

3) Behavioral Models: The Communication Model Describing the Enterprise Data Synchronization Process

For the implemented solution, the data notification exchange follows a very simple path through the hub-and-spoke (or star) integration middleware topology (Figure 3). However, the main challenge that had to be overcome is associating the business entities from one system to business entities in other systems, without introducing direct dependencies between these systems or awareness of other domains or domain-specific identifiers that – semantically – tie these enterprise entities together. For this purpose, an entity correlation service was introduced, using a separate repository of entity IDs that represent logically - or semantically - identical entities across the enterprise. Such correlations will be specified during an initial data setup process by administrative users or via custom automation tools and import/export facilities.

B. Noteworthy Features of the Integration Architecture

Some of the rather interesting features of this real-world integration solution are compiled below, grouped into functional and non-functional characteristics. Several design details are included to impart to the reader some level of context and comprehension of the architectural and technical approaches chosen.

1) Key Functional Attributes

a) Enterprise Data Coherence

Maintaining multiple data copies synchronized, all integrators become symmetrical systems of record for the core/common enterprise data.

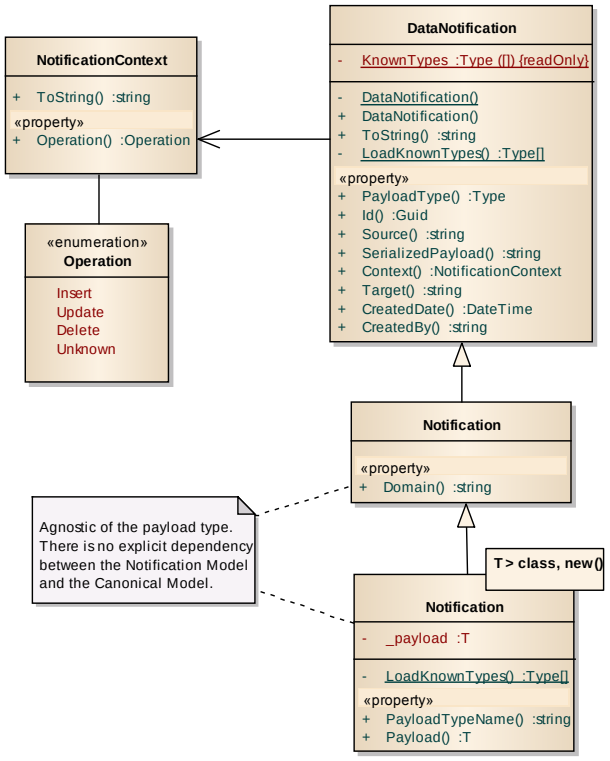

Figure 2. Data notification object model

All systems participating in the integration are able to notify the enterprise about relevant data updates in a particular line of business system without being aware of the other systems that might need this information or of the way in which this data will be consumed.

All systems participating in the integration will be notified of relevant data updates occurring across the enterprise via notifications that encapsulate data payloads following a normalized model. This in turn allows them to keep their own data copy synchronized with the data across the enterprise, while continuing to provision it independently, according to the domain's business rules.

b) Enterprise Functional Coherence

Specialized domain services offered to clients will continue to be managed and augmented within each individual vertical, without the need to cross domain boundaries, since all necessary data is available at the domain level, nearly real-time consistent with the enterprise data.

Decoupled and asynchronous notifications exchanged via the messaging broker keep systems unaware and independent of each other, while allowing the enterprise to grow as needed. Additional applications may be added; if these applications require their own data copy, they will start listening to notifications, and if they also support or require data updates that must be synced with other applications' data sources, then the new applications will also start sending notifications to the broker, to be dispatched and consumed throughout the enterprise, as needed.

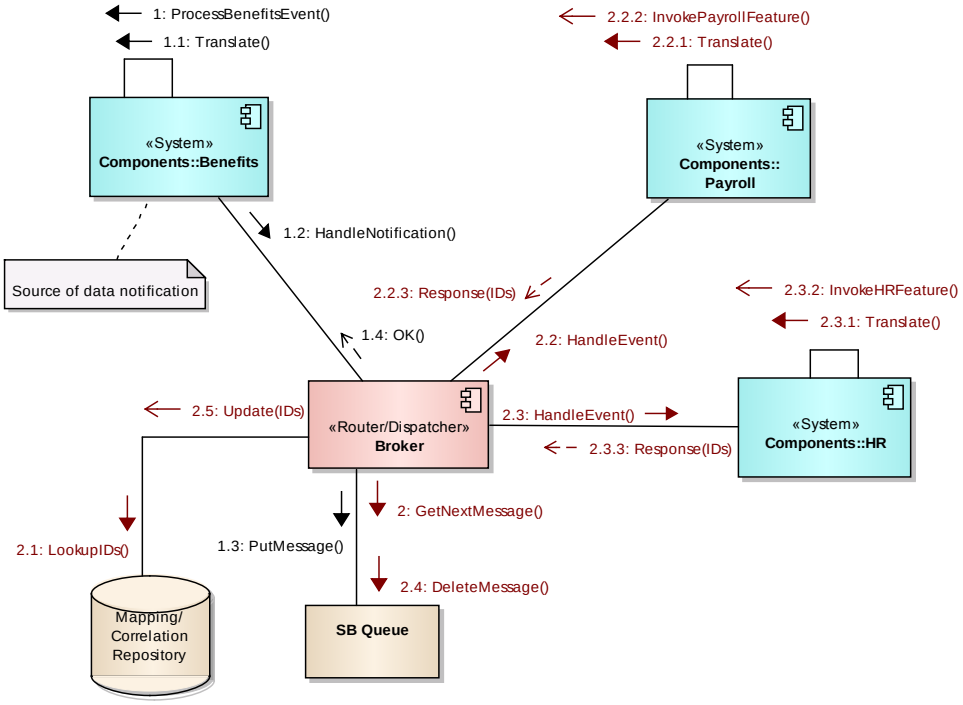

Figure 3. High-level integration communication model mapped to the service broker (star) topology

 ISBN: 978-1-61208-448-0

2) Key Quality Attributes

a) Scalability

Without any architectural changes to the integration framework or the domain systems, new systems can be added to this topology and can be enabled to participate in the integration (assuming they also use their own data source(s) that require continuous or occasional synchronization with the enterprise data). The only two-fold requirement is for these systems to expose a data notification service endpoint to handle enterprise notifications and to be able to raise and react to such data notifications appropriately, while being aware of the canonical model as the lingua franca of the enterprise integration.

b) Testability

Although additional testing frameworks for the integration components must be designed and built, individual systems will continue to be tested independently of each other or the integration middleware.

Components that simulate/generate notification traffic through the integration framework can be built to allow for independent testing of the service broker and the integration infrastructure.

c) Maintainability

The basic SOLID design principles employed, and most importantly the "separation of concerns" (or SoC) principle, ensure a highly maintainable architecture and codebase due to overall high cohesion and low coupling [5] [10].

Domain rules do not escape the boundaries of the system to which they belong, and similarly integration logic is isolated to the broker components and services.

d) High Availability

By employing load balancing and clustering around the integration services and the choice of technology (e.g., Service Bus Farm), the deployment topology was designed so as to ensure high availability as far as the integration components are concerned.

e) Performance

Assuming appropriate technology choices, the integration framework ensures a high throughput of notifications with minimal integration logic (i.e., entity correlation map lookup) required between the moment of receiving a notification and that of dispatching one.

For example, Microsoft's Windows Server Service Bus 1.1 (on premise) can process 20k messages/second (based on 1K message size) with an average latency of 20-25ms [11].

C. Enterprise Integration Patterns Mapping

The integration patterns [7] that were employed in designing and realizing the integration architecture are presented below. They can easily be mapped to the business verticals and integration middleware components as an overlay atop the simplified enterprise system block diagram, as seen in Figure 4.

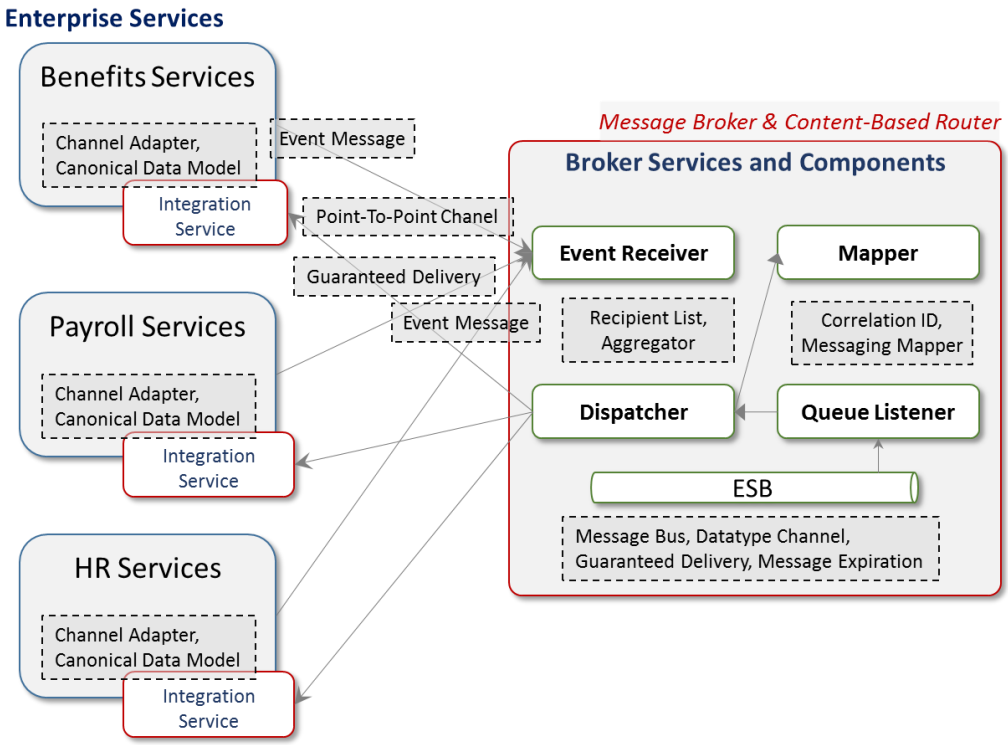

Figure 4. Mapping of enterprise integration patterns to domain systems and to integration middleware components

 ISBN: 978-1-61208-448-0

IV. SUMPPLEMENTARY INTEGRATION MODELS

A. *The Canonical Model's Base Class Details*

The Canonical Model integration pattern [7] has been the central theme of the solution implemented and is the only integration element that was allowed to permeate the enterprise (at each system's integration endpoints). This model can be envisioned as the ubiquitous integration language, which describes entities that are shared across the various domains of the enterprise. However, these entities in turn share data elements that are best modeled separately, as properties on base classes, using elemental inheritance, aggregation, and composition modeling concepts. For the domains in the presented case study, the need to support entity identifiers of different types, active timeframes, and traceability/audit features, led to the design of the model in Figure 5 where all domain entities inherit from the abstract class *EntityBase* shown in the center of the class diagram.

B. *The Canonical Model and the Main Integration Entities*

The main (aggregate root) entities in the integration's lingua franca are Group and Employee. They reflect the primary integration objective: keep Employee and Group demographics data in sync among all enterprise systems, by allowing each system to maintain and operate on their individual copy of the data. The model shown in Figure 6 is specific to the integration solution proposed for the client, aiming at integrating Benefits, Payroll, and Human Resources domains, more specifically for achieving the business goal of cross-selling services to various clients.

Noteworthy here is the fact that if we consider the canonical model as the domain of the integration, then it is following the anemic domain model design anti-pattern [6]. This is because these are simple data containers and do not encapsulate functionality as the integration framework's domain itself is behavior-less. The model's only purpose is to capture and transport data notifications across systems – so, from this (proper) perspective the model is abiding to the Data Transport Object (DTO) pattern of enterprise application architecture [5].

Generic functionality is exposed in the form of service operation contracts for handling notifications (whether a domain system raises a notification or must handle one), but no enterprise features are being implemented here, hence data representation and modeling is of essence and imperatively impacts the success of the proposed system integration solution.

C. *The Enterprise Integration Activity Model*

The overall system integration flow is modeled in the activity diagram in Figure 7, where the various integrating systems and the broker components are bounded by the vertical swim lanes, to indicate where activities and actions cross system boundaries. The diagram also shows how the correlation service is being employed to allow the integration framework to associate the same (logical) clients across domains by looking up and populating the appropriate domain identifiers, as part of the context that wraps the notification data payload passing through the broker.

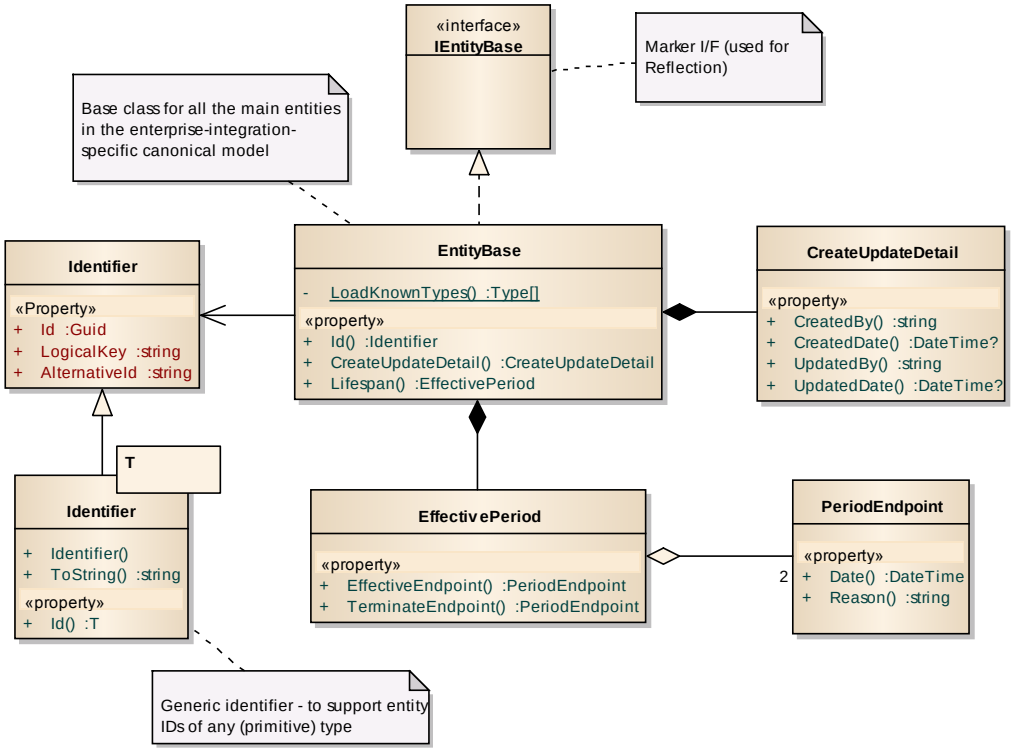

Figure 5. Base class and common elements for the canonical model types

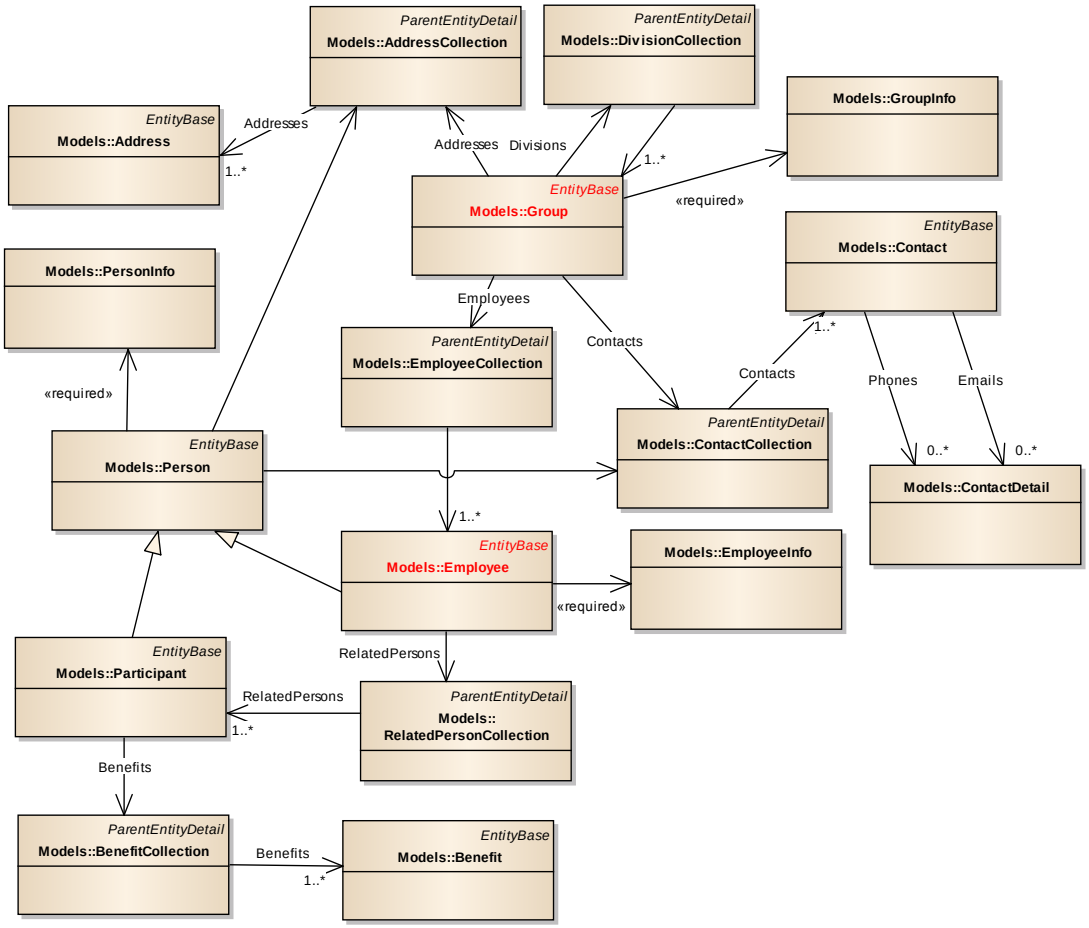

Figure 6. Canonical model's main entities: the payload of the data notifications

Behind the broker services, multiple queues were used as a durable and priority-based messaging mechanism, in order to decouple the various processes that take place at the integration framework level: receiving notifications, processing notifications and their context, and finally dispatching notifications to targeted systems.

V. CONCLUSION

Data integration and synchronization in medium to large multi-domain enterprise systems can be achieved via custom integration frameworks using various enterprise integration patterns and making appropriate technology choices.

This paper presented an actual, real-world integration solution, explained via several structural and behavioral system models, and provided details on how the "maintain data copies" data integration pattern would be realized via a broker-based messaging system. The data exchanged between the various domains is encapsulated inside a canonical model, which is the common data abstraction across the enterprise. This in turn is wrapped inside a context-based, generic, and reusable notification model, allowing systems to react to these notifications based on their own business rules.

The resulting architecture presented here features scalability, extensibility, and high-availability – to mention just a few quality attributes, while supporting near-real-time data synchronization between systems and allowing them to operate without awareness of each other, while using their individual data formats, features, and domain rules.

REFERENCES

[1] T. Erl, "Service-Oriented Architecture: A field Guid to Integrating XML and Web Services," Prentice Hall, 2004.

[2] T. Erl, "Service-Oriented Architecture (SOA): Concepts, Technology, and Design," s.l.:Prentice Hall, 2005.

[3] T. Erl., "SOA Design Patterns," Prentice Hill, 2009.

[4] T. Erl, et al., "Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation," Prentice Hall, 2014.

[5] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley Professional, 2002.

[6] M. Fowler, Martin Fowler. [Online]. Available from: http://www.martinfowler.com/bliki/AnemicDomainModel.html [retrieved: June, 2015]

[7] G. Hohpe, and B. Woolf, "Enterprise Integration Patterns; Designing, Building, and Deploying Messaging Solutions," Addison-Wesley, 2012.

 ISBN: 978-1-61208-448-0

[8] Microsoft, Data Integration. [Online]. Available from: https://msdn.microsoft.com/en-us/library/ff647273.aspx [retrieved: June, 2015]

[9] Microsoft, Functional Integration. [Online]. Available from: https://msdn.microsoft.com/en-us/library/ff649730.aspx [retrieved: June, 2015]

[10] Microsoft, Integration Patterns. [Online]. Available from: https://msdn.microsoft.com/en-us/library/ff647309.aspx [retrieved: June, 2015]

[11] Microsoft, Service Bus for Windows Server Quotas. [Online]. Available from: https://msdn.microsoft.com/en-us/library/dn441429.aspx [retrieved: June, 2015]

[12] M. Iridon, Cândea LLC. "Technical Design Specifications for Enterprise Integration Solution," , 2015, unpublished/internal document.

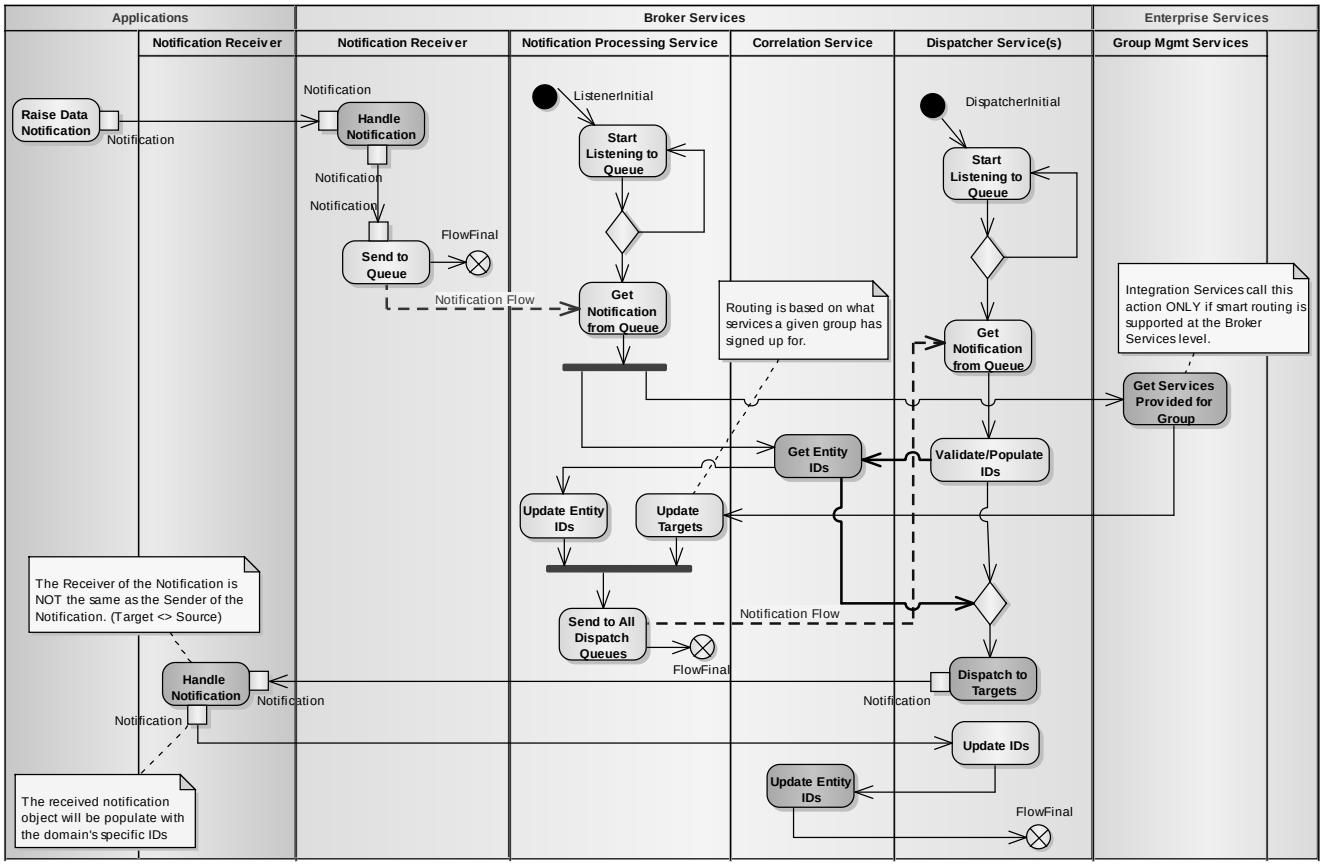

Figure 7. Enterprise integration activity model