Designing a Scalable Social e-Commerce Application

Article *in* Scalable Computing · August 2013

DOI: 10.12694/scpe.v14i2.845

4 authors, including:

Eugenio Zimeo
Università degli Studi del Sannio
125 PUBLICATIONS **994** CITATIONS

SEE PROFILE

DOI 10.12694/scpe.v14i2.845
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 14, Number 2, pp. 131–141. http://www.scpe.org

DESIGNING A SCALABLE SOCIAL E-COMMERCE APPLICATION

EUGENIO ZIMEO[1] AND GIANFRANCO OLIVA, FABIO BALDI, ALFONSO CARACCIOLO[2]

Abstract. eCommerce is gaining a momentum due to the wide diffusion of Web 2.0 technology. Social mining, recommenders and data semantics are moving the focus of eCommerce applications towards context-awareness and personalization. However, the design of these software systems needs specific architectures to support intelligent behaviors, still ensuring important non-functional properties, such as flexibility, efficiency and scalability. This paper proposes an architectural pattern that helps designers to easily identify the subsystems that characterize intelligent enterprise systems. By decoupling transactional behavior from batch processing, the pattern avoids the interference of knowledge extraction and reasoning processes with the state and the performance of the transactional subsystem, so improving scalability. The pattern has been experimented in eCommerce by designing an intelligent and scalable virtual mall.

Key words: Architectural Pattern, Software Systems Design, Scalability, Enterprise Systems, Intelligent Systems, eCommerce

1. Introduction. Enterprise systems represent today an important class of large-scale software that supports many fundamental processes of complex organizations, ranging from resource planning to business intelligence. In this class, eCommerce applications often groups many enterprise assets to offer an integrated and coherent view to merchants and customers for performing business actions.

The adoption of eCommerce as a prevalent channel for selling products is changing the market rules, since competition among vendors is being migrated on the Web. Reaching the widest set of potential customers with information about the commercial products of their interest is one of the challenges that characterize the new solutions and technologies for eCommerce applications.

In the evolution of the Web, Web 2.0 technology represents a milestone with its emphasis on supporting social collaboration and reasoning over data semantics. The new way of interaction is introducing a radical innovation in eCommerce [21], moving the focus towards context-awareness and personalization. If on the one hand, these innovative features improve product selling, they also significantly impact the way modern eCommerce applications are designed and implemented.

Traditional architectures exploited to design enterprise systems (in particular eCommerce applications) need to be revised in order to take into account the desired ability of these systems to "generate" knowledge while working, on the basis of several information sources that could be available as enterprise assets. These sources could be tightly related to the eCommerce application or belonging to different enterprise subsystems that support cross-business features; they can change over time or can be enriched with additional ones when new needs emerge.

Designing an architecture for eCommerce applications in this new scenario is not simple, since designers have to combine the expected intelligent behavior of the system with non-functional requirements, such as flexibility, efficiency and scalability. Separation of concerns, already applied to design complex architectures by separating orthogonal non-functional aspects in insulated modules, could be a viable approach also to design flexible and scalable intelligent enterprise systems.

This paper presents an architectural pattern that helps satisfying both functional and non-functional scalability during the design of intelligent enterprise systems. The pattern decouples the transactional behavior from batch processing, in order to avoid dangerous interference of knowledge extraction and reasoning processes with the state and the performance of the transactional subsystem.

The pattern is applied to the design of a complex eCommerce application, which virtualizes a *mall* composed of a variable number of eShops. The pattern is adopted as a key step of the ADD - *Attribute Driven Design* method [18], which is exploited as a reference process model to design the whole system. The paper shows the effects of the pattern to simplify the comprehension of the system and to improve its design.

The rest of the paper is organized as follows. Section 2 discusses some common architectural patterns exploited to design eCommerce applications and the role of patterns in designing intelligent systems. Section 3

[1]University of Sannio, Dep. of Engineering, 82100 - Benevento, ITALY (zimeo@unisannio.it)
[2]Poste Italiane S.p.A., TI - FSTI - Centro Ricerca, 80133 - Napoli, ITALY (OLIVAG11@posteitaliane.it, BALDIFA3@posteitaliane.it, CARACC52@posteitaliane.it)

presents the *Inference Pattern* proposed. Section 4 discusses the application of the pattern in designing a real system. Finally, Section 5 concludes the paper.

2. Background and related work. Architectural patterns, design patterns and idioms constitute an extensible knowledge base that helps designers to take proper decisions in short times, when they design complex software systems, by reusing solutions already experimented in similar application contexts.

Several architectural patterns have been proposed in the literature for different classes of software systems. In this section, we mainly refer to the patterns that have been successfully exploited for designing enterprise systems and we introduce some recently proposed patterns for intelligent systems, which inspired the one presented and discussed in this paper.

The Treasury Architecture Development Guidance (TADG) [3] proposes some general architectural patterns that can be also exploited to design enterprise systems: *Client-Proxy Server*, *Customer Support*, *Reactor*, *Replicated Servers*, *Layered Architecture*, *Pipe and Filter*, *Subsystem Interface*.

An interesting proposal comes from IBM [1, 2]. It identifies several patterns at different abstraction levels to guide the design of e-Business applications, a specific class of enterprise systems including eCommerce. In this vision, user requirements drive the selection of one or more Business patterns among *Self-Service, Collaboration, Information Aggregation, and Extended Enterprise*.

The first one captures the business interaction between a user and a service, typically a simple web site. The second one enables the collaboration among users. The third one allows users to aggregate data to extract information that is useful for business purposes (e.g. business intelligence). Finally, the fourth one enables a business-to-business interaction to extend the functions of an enterprise with the ones provided by other networked enterprises (e.g. supply chain management).

The business patterns are then used to suggest the adoption of *application* and *integration* patterns, often called *architectural* patterns. The formers capture the functional and non-functional requirements of an application, by proposing the system decomposition in functional and logical subsystems. The latters suggest a way to integrate different subsystems: *Access Integration* patterns define a common entry point for accessing services whereas *Application Integration* patterns represent a way to integrate the flow of actions or the data owned by different subsystems. In the literature, a lot of integration patterns, mainly based on messaging, have been introduced to support Enterprise Application Integration (see [5] for a presentation of these patterns).

A widespread architectural pattern that satisfies self-service interaction model is *Model View Controller* (MVC) [6, 7]. This pattern, with its main variants (*Model View Presenter* [6] and *Presentation Abstraction Controller* [9] are the most known), suggests the organization of the presentation layer and its decoupling from the model of enterprise systems. However, additional patterns are needed to help the design of the other layers proposed by multi-layer and multi-tier decompositions [10]. The layered organization allows for a clear separation of presentation, application logic and data management that multi-tiered architectures exploit in order to partition the different functional components onto dedicated resources, so ensuring important non-functional attributes, such as security, scalability, efficiency and data persistence. In this context, enterprise patterns [4] assume an important role in guiding the detailed design of complex applications.

Service Oriented Architecture (SOA) is becoming the reference high-level architectural pattern to design flexible, reusable and dynamic enterprise and inter-enterprise systems, especially when asynchronous interactions, based on an *Enterprise Service Bus*, are exploited. In [11], the authors propose a mediator-based architecture that decouples service providers from consumers with the aim of binding services on demand on the basis of customers' preferences.

However, the "intelligent" dimension of enterprise systems needs more autonomous and proactive behaviors to satisfy users' needs. Autonomic computing [12] is emerging as a promising approach to include self-* properties in software design and workflow systems [22], whilst MAPE-K (Monitor, Analyze, Plan, Execute and Knowledge) [13] is often used as a reference architectural pattern to design these software systems. This pattern allows for observing the state of a resource in order to intercept possible deviations from a desired behavior that can be controlled by planning adaptations. However, it represents only a starting point for intelligent enterprise systems, where specific patterns are needed to design systems with the ability of inferring knowledge from data generated during the execution.

The paper in [20] focuses on a knowledge management architecture to apply data mining to eCommerce

but it does not address the general architecture of the system. In [14], the authors discuss an architecture to efficiently perform OLAP on the data produced by customers through the interaction with an eCommerce site. Even if, the idea of closing the knowledge loop - users, application, analysis, application changes, users - is similar to the one proposed in this paper, the architecture does not provide users with sufficient hints about the identification of finer-grained subsystems.

In this paper, we present a new architectural pattern, called *Inference Pattern*, that provides software architects with a conceptual framework for designing the business logic of intelligent enterprise systems.

3. Inference Pattern. The pattern will be described according to a typical schema for patterns.

Problem. We want to design a software system able to perform processing on its own data, produced by user interactions, with the aim of expliciting the implicit knowledge that may be useful to users and to the system itself.

Context. Interactive applications generate data from users interactions, which are compliant to models that are typically described through relational, ontological or object-oriented schemas. Data passed during interactions can be processed with the support of data coming from other sources to generate new information that can be useful to understand users' behaviors and preferences. This knowledge, often defined implicit or tacit, can be exploited to personalize the interaction between users and the system, since it eases the knowledge transfer from software to users, by reducing the effort to access data. This improves the effectiveness of the interaction with reference to the business objectives of the system. The knowledge acquired by the system can be in turn reused by other systems.

Solution. The system is decomposed in two logically separated subsystems: the one in charge of answering to user requests and the one responsible of collecting data and preprocessing them. The former becomes passive with reference to the latter that, on the other hand, generates the aggregated data that are useful to improve user interactions and the quality of the data hosted by the interactive subsystem. The logical separation promotes both the knowledge base extensibility, thank to the possibility of using additional systems as subject, and the continuous and concurrent processing with respect to the transactional activity generated by the interactive subsystem, so easing and optimizing the successive deployment.

Participants. In the following, a list of participating subsystems (see Fig. 3.1) is given with a brief description.

- Subject: is the observed subsystem from which implicit knowledge is extracted.
- *Knowledge extraction:* collects the data produced by the interactions between users and the subject and extracts the implicit knowledge.
- *Generation:* generates new knowledge for the subject, by using the implicit knowledge extracted by the Knowledge Extraction (KE) subsystems.
- *Decision Support:* it provides the user with a new knowledge extracted by the KE subsystem to support proper decisions.
- *Recommendation:* suggests information to users. It may use the Generation subsystem to ask for the generation of aggregated data or the Decision Support subsystem to recommend a choice to a user.
- *Personalization:* it personalizes the suggestions with respect to the specific user (or context in general). It is a sort of recommender supporting personalization.

Structural view. In Fig. 3.1, the structural view of the pattern is described by highlighting the participants introduced above and the *use* relationships among them. The red dotted line is a particular use dependency since it, differently from the other ones, works on the Subject by writing onto it. In addition, the view shows two subsystems of KE: *Management* and *Search*, which can be useful to configure the inference rules and to perform particular search operations on the collected data.

The KE subsystem can access the Subject by means of synchronous (dependency regards the external operations of the subject) or asynchronous (dependency is related to the events) interactions.

Related patterns. The proposed pattern is inspired by two known patterns: *Observer* [6] and *MAPE*. The former is a design behavioral pattern that introduces the concept of observing the events generated by a Subject (or Observable system) to drive the behavior of a multitude of Observers. The latter, on the other hand, changes the viewpoint with respect to the subject, which becomes a resource monitored by a manager that is able to analyze the state of the resource and to apply to it, if needed, some state changes (that in turn may produce

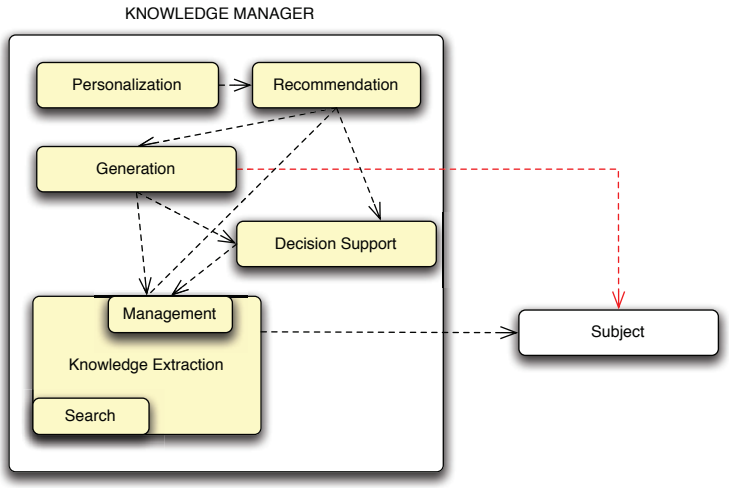

FIG. 3.1. *Structural view of the pattern*

behavioral changes).

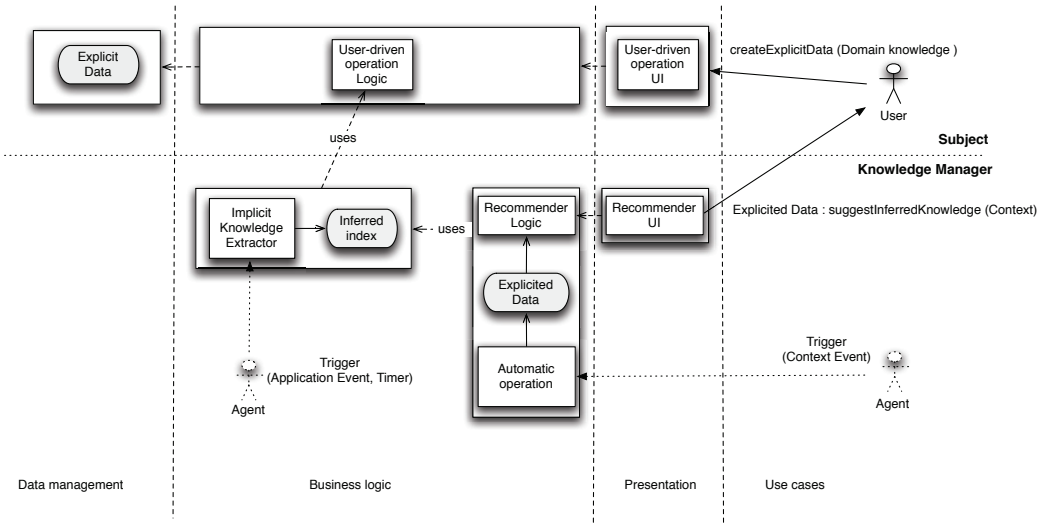

FIG. 3.2. *Partial application of the pattern in a multi-tier context*

The Inference Pattern, differently from MAPE, does not aim at conferring an autonomic behavior to the system but mainly at providing users with an additional knowledge inferred from the data collected from the subject. Therefore, even if, like MAPE, it creates a closed-loop system, the closure is mainly performed by users and, when possible, by the system itself through the writing operations (Generation) applied to the state of the subject. Typically, these operations do not change the behavior of the system, but they make it possible to retrieve further information by users, to feed the knowledge cycle.

Example. Fig. 3.2 shows an example of the pattern application in a typical case that combines interactive transactions derived from users (user-driven operations that store data coming from users' knowledge) and concurrent data extraction and elaborations of (automatic) recommendations to the same user. The (software) agents are in charge of performing the use cases that regard automatic operations. The results of these operations represent the inferred knowledge that is used to provide suggestions in a given context. The figure also shows

the separation of the subsystems in finer grained logical ones, useful to implement a layered architecture that can be easily transformed in a multi-tier one thanks to the allocation of the subsystems onto different resources.

4. Case study: the InViMall system. The Inference Pattern has been exploited to design a scalable eCommerce system, called *InViMall* (*In*telligent *Vi*rtual *Mall*), at Poste Italiane S.p.A.

InViMall is an electronic mall, based on e-community concepts, that enhances the shopping experience of users. It exposes several innovative features through a multichannel interface:

- *Personalized suggestions* [15]. They provide a personalized selection of products based on users interests, preferences, interaction history and the history of related users. Recommenders exploit the relationships between users and products to aid customers in selecting items from a set of choices, suggesting the products fitting their tastes. Users' profiles are either explicitly defined or inferred by analyzing their behaviors during the interaction with the system.
- *Automatic generation of personalized product bundles* [16]. They find the combinations of products that try to satisfy user preferences and requirements, guaranteeing, at the same time, the satisfaction of merchants needs, such as the minimization of the dead stocks.
- *Advanced marketing intelligence* [17]. They propose suggestions to merchants, based on the analysis of sales data from the entire system combined with the profiles of the registered users. They exploit also social mining techniques to create viral campaigns targeted to the most influential members within a community.
- *Faceted Navigation.* They provide customers and merchants with a semantic multi-dimensional search, which is able to reach the desired product by using different attributes as entry point in the catalogue.

TABLE 4.1
Functional areas and potential functional subsystems

Name	Description
eCommerce	eShops mgmt, multi-shop order mgmt, catalogues, shopping list
Model Manager	Semantic catalogue, product/customer relationships
Marketing	Marketing campaigns, analysis of sales and reporting, buyer groups
Social Network	Relationship among users, thematic groups and content sharing
Social Analyzer	Social mining: opinion leader and friendships identification
Selection and Bundle Generator	Prediction rating and automatic bundle generation
Other Systems	Interface to payment, logistics, shipping, and financial systems
Communication	Notifications, e-mails and messages exchange among users
Social Commerce	Products, eShop and bundle ratings, reviews and likes
Personalization	Products and bundle suggestions based on users preferences
Search	Shop and Mall catalogue search, faceted navigation

These functions have to be implemented taking into account typical non-functional requirements for eCommerce systems, such as multichannel access and interoperability with external systems. The multichannel access regards different kinds of devices (e.g. personal computers, mobile devices, kiosks) and communication channels (http, dedicated protocols). Interactions with external systems regard:

- (a) the real-time monitoring of the entire product delivery process across the country, by exploiting the Postal System Logistics platform of Poste Italiane;
- (b) payments, by using any of the several methods offered by Poste Italiane.

4.1. InviMall design. The design of InViMall was performed by following the *Attribute Driven Design* method proposed in [8]. According to this method, it is important to clearly define the requirements, both functional and non-functional with the aim of identifying the architectural drivers.

Therefore, the first phase of the process was the identification of scenarios and use cases that cover the innovative aspects of the system. Hence, some functional areas and possible subsystems were initially identified (see Table 4.1)

Due to the absence of a reference architectural pattern for the business logic of this kind of systems, the

next step was the identification of dependencies among subsystems, by exploiting use cases analysis, with the aim of building a *Dependency Structure Matrix* (DSM) [19]. It was useful for identifying and analyzing the mutual dependencies among the subsystems.

As Fig. 4.1 shows, reasoning only about the functional areas created a dependency graph with several mutual-dependencies, so generating a lot of coupling in the systems that could negatively impact flexibility, reusability, performance and scalability.

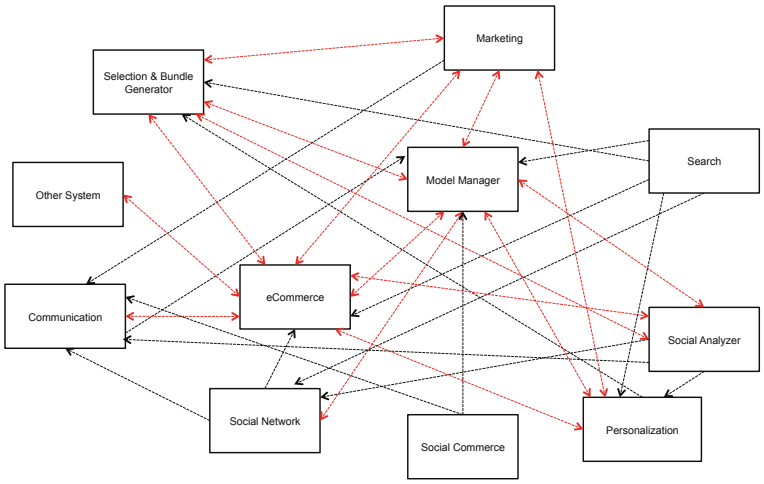

FIG. 4.1. *Preliminary decomposition of the InViMall system in subsystems*

4.2. Application of the Inference Pattern. The ADD method, to design complex software systems, suggests to identify the architectural drivers from the software requirements with the aim of selecting the proper architectural patterns to adopt as a reference guide for the whole design phase. This allows software architects to reuse the consolidated experience as an existing knowledge base to reduce the design effort.

In the case of complex and innovative software systems, existing patterns could not be sufficient to identify subsystems and their relationships, so limiting the benefits of the ADD method. We encountered this problem during the beginning of the design phase of InViMall, due to the lack of specialized patterns for designing the business logic of intelligent eCommerce applications. To overcome the problem, the architectural drivers of the InViMall system were generalized and an effort was done to abstract a solution by defining the architectural pattern presented in the paper.

The application of the pattern caused a complete refactoring of the design (see Fig. 4.2), introducing a strong reduction of mutual dependencies and a clear separation between the interactive and transactional subsystem (called *Mall*), and the batch subsystem (called *Intelligent Mall*).

The former groups the typical subsystems that characterize an electronic mall, extended with social network features. The latter hosts (i) a KE, which implements knowledge reasoning, predicted rating features and social mining, (ii) a decision support subsystem that enables merchant to configure and manage innovative marketing campaigns, (iii) a *Generation* subsystem that works onto the state of the Mall to store new explicit knowledge extracted by KE from the implicit one collected from several sources. It is worth to note that several features were tangled among different subsystems before applying the pattern, so generating mutual dependencies and coupling.

The two main subsystems derived from the decomposition have different characteristics that impact their implementation. The Mall is a typical transactional system and therefore it is characterized by concurrent accesses, data persistence and ACID transaction management. On the contrary, the Intelligent Mall is mainly a batch system using background processes to manipulate large amounts of data for analytical processing and to generate correlations among entities. Due to these peculiarities, non-relational persistence systems (e.g.

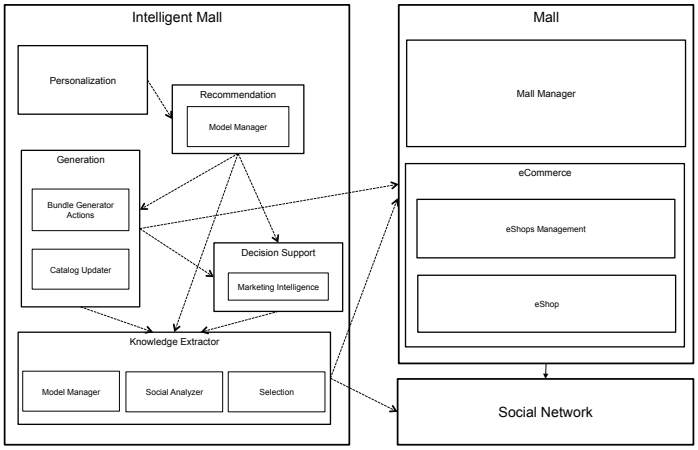

FIG. 4.2. *InViMall architecture after the application of the pattern*

columnar or graph-based DBs) could be exploited to improve the performance of the whole Knowledge Manager.

The derived architecture allows for adding innovative features without impacting the transactional subsystem and makes it easy the integration of off-the-shelf software. Moreover, it enables the reuse of single subsystems or groups of them in different business contexts of an enterprise. Finally, the pattern suggests a way to exploit the implicit knowledge not only to suggest recommendations to users but also to drive automatic actions (from the Intelligent Mall to the Mall) that aim at satisfying the business objectives of the enterprise.

Particular attention should be given to the proper allocation of business use cases to each subsystem, especially those ones that extend or include use cases scattered across Mall and Intelligent Mall. This integration, which is specific to the application, may be performed at the control layer trying to avoid strong coupling and synchronization. To this end, an event-driven control layer is suggested to capture the events coming from the user interface and to propagate them to the specific subsystems, so preserving a high degree of concurrency and low degree of coupling.

The integration between the two subsystems could be performed by exploiting one or more of the following approaches: (a) Hard Coded integration; (b) Extract, Transform, Load (ETL) techniques; (c) Data Virtualization technologies; (d) Enterprise Service Bus (ESB).

The simplest approach (in term of learning curve) is based on the hard coded implementation of the KE that periodically accesses to the functional interface exposed by the transactional system. Several problems need to be taken into account: timing policies (a short period causes continuous access to the transactional subsystem; long period causes potential inconsistency); the list of functions to be invoked (changes or additions of the features in the Subject would require the revision of the source code of the KE subsystem).

ETL consists of three steps: 1) data extraction from heterogeneous data sources; 2) data transformation for the target; 3) data loading into target. This technique assumes the inconsistency between the data source and the destination be tolerable, but this is not true in modern eCommerce applications since they need providing users with up-to-date information through recommenders.

Data virtualization allows for accessing heterogeneous data systems providing a global integrated view. This technology uses functional interfaces or data access as abstract sources through specific connectors (ODBC, JDBC, REST, SOAP, etc). This integration allows for real-time data accesses but may generate access conflicts with the transactional sub-system. The problem can be alleviated by using caching techniques that reduce the number of concurrent accesses towards the transactional subsystem, keeping the already retrieved data in a cache, so ensuring significant performance gains.

The integration through ESB allows for a strong decoupling and a reuse of single components. This technology provides a messaging system that is able to asynchronously exchange information between publishers

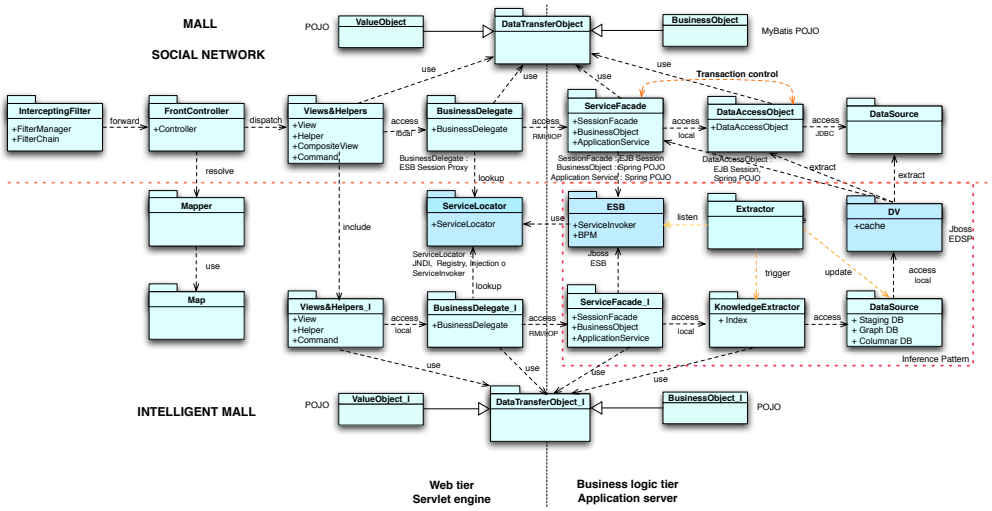

FIG. 4.3. *Integrating the Inference Pattern with the enterprise patterns*

(producers) and subscribers (consumers), so improving concurrency and reducing coupling.

Since ESB is typically not exploited for massive volumes of data, we decided to combine ESB, to communicate events, with (caching-based) data virtualization, for retrieving big data. This solution enables both real-time access to transactional data and loose coupling among the sub-systems, ensuring, at the same time, a strong reuse of services.

Fig. 4.3 shows the integration of the architectural pattern proposed in this paper with the well known enterprise patterns [4] typically exploited to design applications according to multi-tier architecture design.

Following this architecture, software components are classified on the basis of their possible deployment, so identifying two main classes: (a) Web and (b) Business logic components. The application of the Inference Pattern splits the packages in two vertical groups (Mall and Intelligent Mall) and identifies the main packages involved in its implementation (red dotted box).

4.3. Technical architecture and deployment. Following the structure view shown in Fig. 4.3, the implementation of the application has been performed by exploiting specific technology that already offers mechanisms that ease the implementation of the enterprise patterns.

The presentation layer has been developed with GWT (Google Web Toolkit), whereas the business logic has been developed with Spring Framework and MyBatis to interface a MySQL Database. Almost all of the functional components are invoked by Enterprise Java Beans. The data Layer has been built upon an RDBMS (MySQL server), whereas a non-relational persistence system (Neo4J Graph DBMS) has been exploited to improve the performance of the Social Analyzer hosted by the Knowledge Extractor.

Fig. 4.5 shows the technical view of the InViMall architecture. This view points out the logical subsystems (organized in layers) and the kinds of servers and frameworks used to develop the platform. The presentation logic exposes the business functions through HTTP, exploited by both browsers and apps.

The data of the Mall subsystem are shared with the Intelligent Mall subsystem through a staging database, implemented by using the data virtualization mechanisms offered by JBoss Enterprise Data Services (Teiid). It allows for accessing heterogeneous data systems providing a global integrated view.

The implementation of data virtualization has been built on three levels (see Fig. 4.4). The bottom level represents the physical layer that contains the databases of Mall Manager, Social Network and the web services exposed from the legacy eCommerce subsystem adopted (Magento). The middle level abstracts the data sources with one to one mapping from DB to web services data. At this level, a data type transformation function is used to improve the accuracy of data type definition. The top level is composed of a view of the different data sources which are incorporated in a global view to represent the virtual database. This is used only for

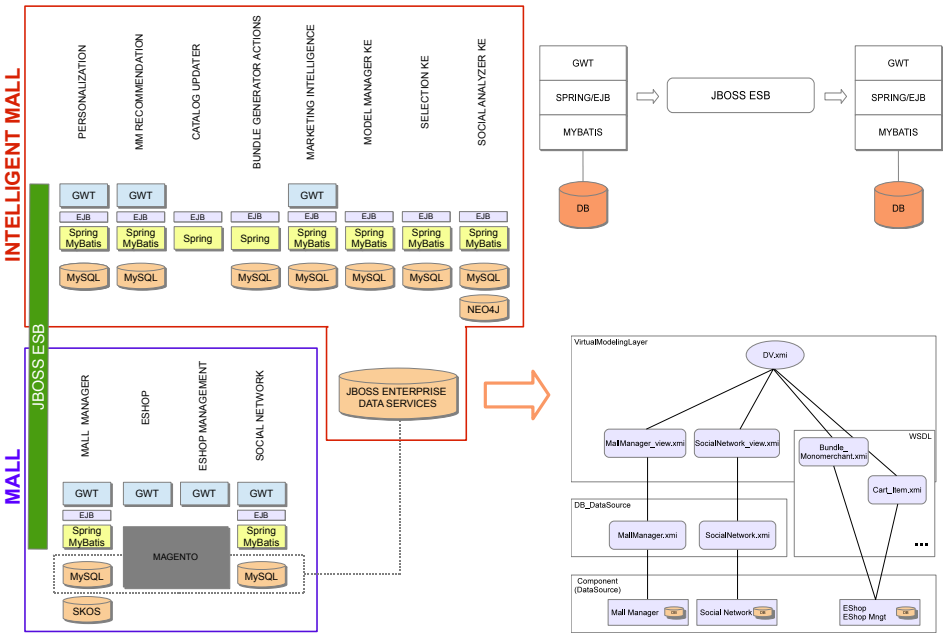

FIG. 4.4. *Technological view: components, data virtualization configuration and communication*

retrieving data while any insert/update/delete operation is performed through the ESB services that wrap the mall functions.

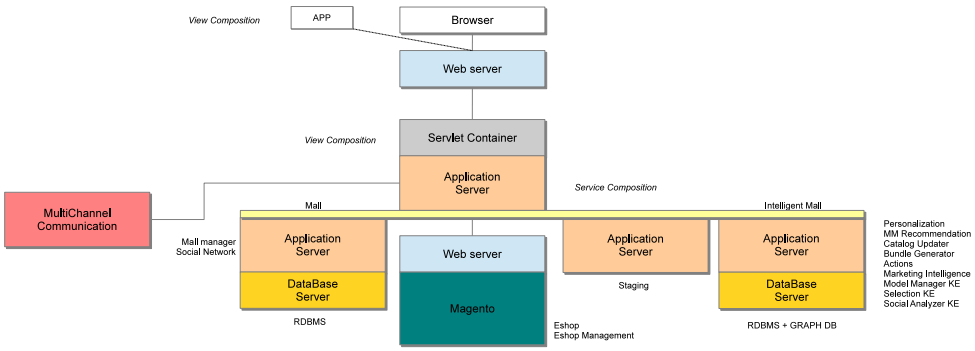

FIG. 4.5. *Technical view: hosting environments*

The components of the InViMall system expose a stateless Session Bean for each interface, using a DAO Layer (Data Access Object) based on MyBatis 3.1.1 for the access to the database. The communication between the components is performed through the JBoss ESB (see Fig. 4.5 (b)).

The whole system has been deployed on virtual machines that are placed on four different virtual LANs. The technological stack used allowed for generating several software deployment packages: a .war artifact for the integrated web tier; an .ear artifact for each functional subsystem implemented through EJB-Spring-MyBatis components; a .vdb artifact for the virtual database; an .esb artifact for packaging the services connected with the ESB. In particular, according to the Inference Pattern, the developed artifacts were deployed onto the virtualized system architecture shown in Fig. 4.6 and organized as follows:

- vLAN D0 (Static presentation tier) hosts an Apache web server for HTTP dispatching and static HTML

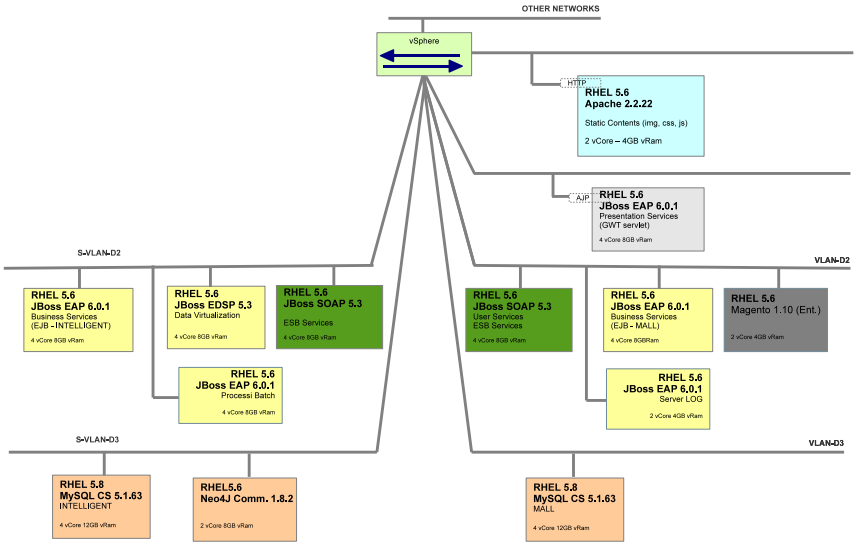

FIG. 4.6. *Application deployment*

pages handling;

- vLAN D1 (Dynamic presentation tier) hosts an Application Server for the integrated GWT Servlets;
- vLAN D2 (Business logic tier) is switched in two segments and hosts: on the right side, an application server for the ESB, an application server that runs the Mall business logic, a log server, and Magento; on the left side, an application server for the ESB, an application server that runs the Intelligent Mall business logic, an application server to host the data virtualization layer, and an application server that runs the batch processes;
- vLAN D3 (Data management tier) is also switched in two segments and hosts: on the right side, a MySQL DBMS for storing the Mall data; on the left side, a Neo4J Graph-based DBMS for storing social data, and a MySQL DBMS for storing data for recommendation and decision support extracted from the virtualized data base or from the Graph-based DB through batch processing.

It is worth to note, that the deployment analyzed before is a simplification of the real deployment, which considers also the replication of each tier to address reliability and horizontal scalability through replication.

5. Conclusion. The paper presented an architectural pattern that helps software architects to design intelligent enterprise systems that combine interactive features with batch processing to infer knowledge from large amounts of data.

The pattern was applied with success to design a complex application that implements an electronic mall. Thanks to it, software architects clearly and rapidly identified the functional subsystems, by associating to them use cases. It, also, contributed to create a loosely coupled system with a clear logical separation between the transactional and intelligent sub-systems, and a further separation among the subsystems of the intelligent side.

The benefits of the pattern were discussed also with respect to the integration techniques that can be exploited to efficiently implement the pattern. The possibility to deploy the subsystems onto different resources with specialized characteristics makes the overall system efficient and flexible, since additional subjects can be easily added. Scalability is achieved by properly deploying transactional and batch oriented subsystems onto different hardware resources linked to switched network segments.

The pattern enables the integration of both real-time, event-driven interactions, and data accesses for large volume of data. In the future, the application of more sophisticated integration middleware platforms that transparently combine the two benefits above will be investigated [23].

Acknowledgements. This work was partially supported by Italian Ministry of Economic Development in the context of the Intelligent Virtual Mall (InViMall) Project MI01-00123.

REFERENCES

[1] http://www.opengroup.org/togaf/
[2] http://www.ibm.com/developerworks/patterns/
[3] http://www.opengroup.org/togaf/
[4] http://martinfowler.com/articles/enterprisePatterns.html
[5] Hohpe G., Woolf B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison Wesley (2003)
[6] Gamma E., Helm R., Johnson R., Vlissides J.: Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley (1995)
[7] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M.: Pattern-Oriented Software Architecture - A System of Patterns, John Wiley & Sons, (1996)
[8] Potel M., MVP: Model-View-Presenter: The Taligent Programming Model for C++ and Java. (1996)
[9] Plakalovic D., Simic D.: Applying MVC and PAC patterns in mobile applications. Journal of Computing, 2(1) (2012)
[10] Fowler M. J.: Patterns of Enterprise Application Architecture. Addison Wesley (2002)
[11] Cong S., Hunt E., Dittrich K. R.: IEIP: An Inter-Enterprise Integration Platform for e-Commerce Based on Web Service Mediation. In: European Conference on Web Services - ECOWS, pp. 201-210 (2006)
[12] Kephart J., Chess D.: The vision of autonomic computing. Computer, 36(1):41-50, (2003)
[13] Huebscher M. C, McCann J.A.: A survey of autonomic computing - degrees, models, and applications. ACM Computing Survey, 40(3):1-28 (2008)
[14] Ansari S., Kohavi R., Mason L., Zheng Z.: Integrating E-Commerce and DataMining: Architecture and Challenges. In: International Conference on Data Mining, IEEE, pp. 27-34 (2001)
[15] Birtolo C., Ronca D., Armenise R.: Improving accuracy of recommendation system by means of Item-based Fuzzy Clustering Collaborative Filtering. In: 11th International Conference on Intelligent Systems Design and Applications - ISDA '11, IEEE, Spain (2011)
[16] Birtolo C., De Chiara D., Ascione M., Armenise R.: A Generative Approach to Product Bundling in the e-Commerce domain. In: the 3rd World Congress on Nature and Biologically Inspired Computing - NaBIC'11, pp. 169-175. IEEE, Spain (2011)
[17] Birtolo C., Diessa V., De Chiara D., Ritrovato L., Customer Churn Detection System: Identifying customers who wish to leave a merchant. In: 26th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems, LNCS Springer, Amsterdam, June 17-21, pages: 411-420 (2013)
[18] Bachmann F., Bass L.: Introduction to the Attribute Driven Design Method. In: the 23rd International Conference on Software Engineering - ICSE (2001)
[19] Clements P., et al.: Documenting Software Architectures: Views and Beyond. 2nd edition, Paerson Education (2011)
[20] Yu H., et al.: Knowledge Management in E-commerce: A Data Mining Perspective. In: Management of e-Commerce and e-Management, IEEE (2009)
[21] Guo S., Wang M., Leskovec J.: The Role of Social Networks in Online Shopping: Infor-mation Passing, Price of Trust, and Consumer Choice. In: Conference on Electronic Commerce, ACM (2011)
[22] Polese M., Tretola G., Zimeo E.: Self-adaptive management of Web processes. In: Web Systems Evolution (WSE), IEEE, pp. 33-42 (2010)
[23] Zagarese Q., Canfora G., Zimeo E., Alshabani I., Pellegrino L., Baude F., Efficient data-intensive event-driven interaction in SOA. In: Symposium on Applied Computing - SAC 2013, ACM, 1907-1912, (2013)

Edited by: Viorel Negru and Daniela Zaharie
Received: June 2, 2013
Accepted: Jul 9, 2013