Adaptive Data Dissemination and Caching for Edge Service Architectures built with the J2EE

Erich Liebmann
Distributed Systems Group
Information Systems Institute
Vienna University of Technology
Argentinierstrasse 8/184-1
1040 Vienna, Austria
+43 676 9455733
e9625216@student.tuwien.ac.at

Schahram Dustdar
Distributed Systems Group
Information Systems Institute
Vienna University of Technology
Argentinierstrasse 8/184-1
1040 Vienna, Austria
+43 1 58801 18414
dustdar@infosys.tuwien.ac.at

ABSTRACT
The deployment of distributed enterprise applications and e-business solutions, that leverage edge service architectures across wide area networks, require flexible and adaptable models for data dissemination and caching. In this paper we present the design of a framework built on top of the Java 2 Enterprise Edition that provides proactive data dissemination and caching for distributed e-commerce solutions. The utilization of an adaptive push and pull approach combined with the integration of business logic into data dissemination and caching decisions leads to a communication infrastructure that enables business participants to communicate efficiently. The association with the problem domain allows us to take advantage of application specific semantics and streamlines the extension of the architecture with additional business relevant services.

Categories and Subject Descriptors
H.3.4 [**Information Systems**]: Information Storage and Retrieval – Systems and Software – *Distributed Systems;* D.2.11 [**Software**]: Software Engineering – Software Architectures – *Domain Specific Architectures, Data Abstraction;* C.2.4 [**Computer Systems Organization**]: Computer Communication and Networks – Distributed Systems – *Distributed Applications, Distributed Database*

General Terms
Design, Performance

Keywords
Data Dissemination, Caching, Edge Services, Adaptive, Push, Pull, Distributed Enterprise Applications, Data Services Layer, J2EE, JMS

1. INTRODUCTION
The increasing number of e-business solutions providing global services result in widespread use of the edge services architecture that minimizes network traffic and increases response times and availability. The resulting business collaborations pose new challenges in data dissemination and caching among business participants distributed around the globe. While traditional three tier architectures come with a tight coupling of the data tier and the business tier, today's distributed enterprise applications often need to access data across the internet or other wide area networks. The edge service architecture is a typical scenario introducing the delegation of business logic and the distribution of business data across multiple servers. Edge servers deployed topologically close to client applications handle simple business logic tasks to increase overall performance and availability. More sophisticated business processing is delegated to the central backend system that is directly connected to the data center. Despite the ability to delegate business processing to the backend system, edge servers require automatic notification of data changes to preserve temporal coherency and to be able to react to changing conditions.

Modern multi-tier enterprise application development platforms such as the Java 2 Enterprise Edition provide a powerful programming environment for three tier enterprise applications and extend it with middleware support and web services to facilitate the communication with business participants. Furthermore, many object-to-relational mapping tools [28,38,39] exist that provide caching support for conventional three tier architectures and clustered environments. The distributed nature of the edge service architecture combined with the communication latency of wide area networks and the costs involved in transferring information across the internet pose the requirements for more efficient data dissemination models for modern enterprise applications. Complete data replication is not suitable for the edge service architecture and can be partly substituted by more adaptive and flexible data dissemination and caching techniques.

With the evolution of the internet and the increasing number of large scale distributed systems a number of mainly theoretical papers [29,32,23,20] have evaluated flexible data dissemination and caching techniques. Most adaptive approaches suggest the utilization of both push and pull for data propagation. Depending on the algorithm chosen and the application data used, the system

will favor push or pull to minimize the number of data items transmitted. Further research [33,16,19] into data dissemination both in large scale and asymmetric communication environments highlight the benefits of server initiated and client initiated broadcast and caching models. A transfer of data dissemination technologies developed for traditional web applications to modern enterprise applications and the edge services architecture could significantly improve data propagation among business participants.

In the past few years numerous experts in the field of distributed systems have published blueprints, best practice advices, and design patterns to aid developers in creating architectures that are flexible enough to meet changing business requirements. Our design of the data dissemination and caching framework on top of the Java 2 Enterprise Edition can guide software engineers and system architects in creating solutions for edge service architectures that adequately handle data exchange and caching.

The main emphasis of our work is on providing an architectural design that allows the seamless integration of adaptive data propagation into existing distributed enterprise applications by leveraging the technologies provided by the Java 2 Enterprise Edition. The remaining discussion is tailored to explain integration issues. Algorithm details and system measurements are beyond the scope of this paper.

The framework presented in this paper combines speculative data dissemination with caching in order to minimize data storage requirements and maximize temporal coherency. The knowledge of both edge servers and backend systems is used to decide which data items are proactively disseminated and which data items will be cached. Depending on the algorithm used and the data operated on, the system's behavior will be similar to pure caching or pure replication. We believe that caching can significantly reduce the storage requirements at edge servers and that adaptive and proactive dissemination can reduce the amount of network traffic transferred across wide area networks between the edge servers and the backend system.

Our work is further motivated by those in the research community who claim [17] that we require adaptable middleware solutions that change according to application requirements and resource availability. Our framework allows leveraging application specific semantics to optimize the dissemination and caching behavior.

The remainder of this paper is organized as follows: Section 2 introduces the edge service architecture and explains how the framework can minimize data storage requirements, minimize the amount of data transferred, and maximize temporal coherency. Section 3 describes the technologies utilized and outlines the design and implementation that lead to a flexible and extensible integration into existing enterprise applications. A detailed discussion including a comparison with related approaches and the areas of ongoing research is presented in section 4.

2. EDGE SERVICE ARCHITECTURE

The nodes in a general distributed information system can be classified into *data sources*, which provide the base data to be disseminated, *clients* which are net consumers of information, and *information brokers* that mediate data from the data source to the clients [24]. In distributed enterprise systems data is combined with business logic that performs tasks relevant to the application domain. In a general edge service architecture the data source is substituted by the central backend service that performs critical business tasks and is directly connected to a data center. The edge servers substitute the information brokers and perform business tasks partly in collaboration with the backend system. The edge service architecture facilitates distribution of both business data and business logic and empowers enterprises to offer scalable, highly available e-commerce solutions with optimized response times.

Propagation of data can be achieved by pushing or pulling data either periodically or on demand using the point-to-point or publish-and-subscribe model. We will combine the benefits of push and pull and apply it to the edge service architecture. The diagram in Figure 1 shows a typical edge server architecture that provides the base for the integration of our adaptive data dissemination and caching framework. The propagation techniques (pull-on-demand, speculative push and notifications) introduced by the framework and shown in the diagram will be described in detail in the following sections. The central backend system and the individual edge servers exchange information in order to service client applications. Both edge and backend servers are equipped with an application server that enables flexible presentation and business logic to be applied to the data propagated among individual nodes. The backend cluster is directly connected to a database or a data center providing the business data for the entire enterprise application. Each edge server is equipped with a cache that is maintained in a relational database.

2.1 Data Replication and Caching

It is evident that the performance of each individual edge server is best if all data required for business tasks is cached and therefore locally available. We believe, however that internet based data replication is not applicable and hardly feasible in the presented scenario. Data replication results into transmission of every data item to all edge servers and requires processing and storage resources to handle the increasing data volumes of current business solutions. As highlighted admittedly in a different context [18] combining push and pull might lead to superior solutions than pushing everything to the already overloaded edge servers.

The shortcomings of a data replication solution raise the question if a simple data cache at each edge servers would be sufficient. In a conventional caching system the client tries to locate the required data items in its local cache and upon cache miss contacts the appropriate data source. It is important to note that this client initiated technique doesn't provide the data source with the ability to propagate notifications of modified and added data items. We assume, however that a large number of business applications require notifications of newly added data items to update product presentation and to adapt its business logic to the changing conditions. Furthermore it is crucial to guarantee that business tasks are performed on the most recent data set.

For the remaining discussion we assume that internet based replication and conventional caching is not suitable for the application scenario discussed.

2.2 Adaptiveness for Edge Services

The adaptive data dissemination and caching framework tries to provide a communication infrastructure that overcomes the shortcomings of both replication and caching by transferring specula-

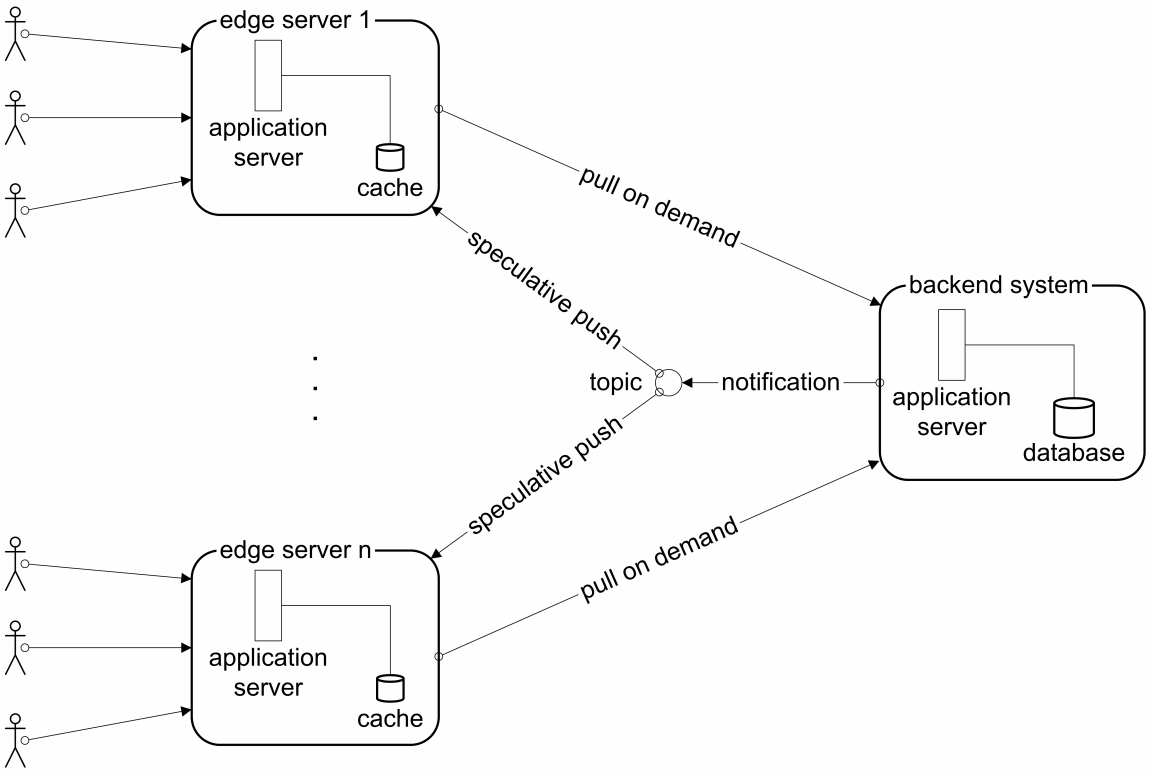

Figure 1 Edge Service Architecture

tive data dissemination and caching techniques to the Java 2 Enterprise Edition.

We will use a relaxed consistency model together with application specific semantics and data characteristics to provide an efficient Data Service Layer.

The applicability of the framework depends on the application data, typical data access patterns, workload characteristics and the data requirements of typical business tasks. In general the framework is designed for large data records that are centrally written (at the backend system) and propagated to multiple locations (edge servers) for subsequent reads. Supporting writes at multiple locations is possible with the presented framework either by delegating write operations to the backend system or by introducing locking or reconciliation techniques. It should be noted that the relaxed consistency inherent in the presented framework requires sophisticated solutions for distributed modification operations.

The presented framework takes advantage of the fact that most applications require a small subset of the total available data to perform most business tasks. We distinguish between operational data (frequently used) and archive data (infrequently used). The distribution of operational data allows us to outsource data access and business processing to the edge servers. With the architecture outlined it is possible to process business tasks that use operational data locally at each edge server while delegating archive intensive business tasks to the backend system. The decision if a given business task should be delegated to the backend system can be determined either at design time relying on application semantics or at runtime depending on the current number of cached items required for the particular processing request. We are currently investigating if the presented framework can be combined with distributed e-service workflow [14] to improve collaborative business process management [25, 30, 12] in edge service architectures unleashing the distribution of even more complex business tasks. In particular with a context aware, proactive delivery of task specific information [1] we could achieve interconnectivity that solely transmits data required by the executing tasks at each edge server.

We identified three main areas of application for the framework presented namely news propagation systems, applications with access patterns typical to monitoring applications, and e-commerce applications. The following discussion explains the framework in the context of an e-business application.

Generally, data provided and processed by e-business applications can be classified into a product catalog, orders, user profiles and inventory information. The TPC-W [40], a transactional web e-commerce benchmark, distinguishes between several scenarios (workload mixes) for a general e-business application.

The product catalog is used to maintain information on the products offered by the e-commerce application. It can be modified at the central backend system and is accessed by clients at individual edge servers. Our proposed framework perfectly fits the requirements for product catalog dissemination to and caching at edge servers. The structure of data used for a typical product catalog together with the workload characteristics described in the TPC-W standard and experienced in real world scenarios allows us to minimize data storage requirements at edge servers while maximizing temporal coherency.

We will apply the framework to disseminate a typical product catalog of an e-business application. The relaxed consistency inherent in the presented framework requires order processing to be delegating to the central backend system. User profile management on the other hand is better achieved by using data replication techniques that introduce relaxed consistency by leverage locality and frequency and rely on reconciliation techniques to resolve data conflicts [21]. The design of the framework streamlines the integration of order delegation and reconciliation techniques resulting into a consistent interface for typical e-business applications.

It might be argued that the optimization of product catalog storage and dissemination requirements represents only a small part of an entire e-commerce application and will not impact overall performance. It should be considered, however, that in contrast to order requests and user profiles, the storage requirements for a product catalog including product description, manuals, and illustrations are immense. Furthermore, it is known from real world experience and the TCP-W standard that in a typical e-commerce application between fifty and ninety-five percent of client requests result into product catalog browsing.

To apply the framework to the product catalog example, we define a product index, and product details. The product index provides a short description of each product while the product details contain the manuals, description, and illustrations.

2.3 Collaboration

The communication pattern introduced by the framework is shown in Figure 1 Edge Service Architecture. In order to (a) reduce the data storage requirements at each edge server and to (b) minimize the amount of data transferred across the wide area network, we restrict update propagation to the product index. A dispatching algorithm that closely works together with the business logic is responsible to determine, if product details should be proactively disseminated together with the index. This is especially useful in application scenarios where product items required by edge services are known in advance or can be predicted from previous requests. For e-commerce solutions we could leverage the geographically varying popularity of products and browsing statistics to decide which product details should be pushed proactively. A number of research projects [13,2] in diverging problem domains have shown that proactively pushing data can result into reduced network traffic and increased performance and scalability.

Each edge server is equipped with a relational database that is used to store the product index and cached product details. The index is updated each time the edge server receives a notification from the backend system. In case the edge server receives a proactively pushed data item, a caching algorithm that closely works together with the business logic of the edge service decides if the data item should be stored in the cache.

In many cases an individual edge server may not be interested in or may not have a contract for all product items provided by the backend server. In such cases it is possible to add transparent filtering at the middleware layer that decides which edge server is going to receive which data items.

The edge servers rely on a full product index to present the product catalog to clients. Furthermore it enables local queries on all products as long as the search criteria use information available in the product index. If a specific data item is required and can not be found in the cache the edge server issues a pull request and retrieves the data item directly from the backend service. Once again the caching logic may decide if the newly retrieved data item is only used for the current business request or put into the cache for future tasks.

In scenarios of edge servers with similar data requirements the utilization of data broadcasting on demand [19] could further reduce network load. To integrate broadcasting functionality into the presented framework we would simply instruct the backend service to broadcast data requests to all edge servers instead of replying directly to the initiating node.

An expiration technique at each edge server is responsible for removing items proactively pushed or actively pulled into the data cache. The expiration algorithm can be influenced by the business logic or rely on query usage statistics to decide which data items should be deleted.

2.4 Consequences

The presented data dissemination and caching model provides several benefits. We can achieve a maximum temporal coherency and therefore a reduction of cache staleness by automatically dispatching notifications on product catalog changes. Furthermore edge servers are not required to constantly poll backend services to keep the product index consistent. In systems with a large number of edge servers we can significantly reduce the amount of traffic by avoiding polling behavior.

The number of data items transferred and cached can be minimized since both the backend system and each edge server determine which items are proactively disseminated and cached. It remains to be evaluated if the introduction of broadcasting on demand (as discussed above) can further reduce the network load for the suggested communication infrastructure.

In application scenarios with predictable data usage we can proactively fill caches, therefore increasing the performance of most business tasks running at edge servers. Furthermore the one-to-many distribution significantly reduces the amount of data propagated from the backend system to the edge servers.

The fact that a full product index is available to each edge server together with the decision to put the cached data into a conventional relational database make it easy to integrate the data received from the backend system with information originating form other business partners or generated by the edge server itself.

The integration of the business logic and the dissemination framework allows edge servers to delegate data intensive operations (such as a full text search of the product details) to the

backend system. Instead of transmitting all data records required by a particular task the backend system would perform the delegated task and return the result back to the edge server that mediates it to the client. This allows us to process regular tasks directly at the edge servers while delegating tasks that rely on a complete copy of the archive back to the backend system. As outlined above the decision which business tasks are performed at the edge servers and which processing requests are delegated to the backend system can be made either at design time or at runtime depending on the number of missing data items available in the cache. Therefore the framework can easily be used for distributed applications that have data access patterns that are typical of monitoring applications.

With the suggested architecture the data dissemination and caching behavior is completely transparent to the business application and should in no way impact the business logic that can be performed on top of the caching layer. The following section describes the integration of the adaptive data dissemination and caching framework into the edge service architecture. We present an application agnostic architectural design that enables the integration of adaptive techniques for data propagation and caching into existing enterprise applications. The framework provides well defined extension points that are used to integrate application specific algorithms for data dissemination and caching that are tailored to a particular enterprise application. Algorithm details and system measurements need to be evaluated in the context of a concrete application (domain), which is beyond the scope of this paper.

3. ARCHITECTURE

We are now faced with the challenging task of integrating the adaptive data dissemination and caching framework into the edge service architecture. The Java 2 Enterprise Edition provides us with the necessary building blocks such as message oriented middleware, remote method invocation, and container managed persistence, while the adaptive push and pull approach originating from traditional web based applications provides us with the theoretical foundation. This combination enables the integration of business logic into data dissemination and caching decisions.

3.1 Technologies

We utilize a J2EE [15] 1.3 compliant application server that provides a reliable platform for the business processing logic and supports container managed persistence, resource pooling, transactions, and caching for local database access. This enables us to concentrate on data dissemination aspects rather than on the seamless integration of the local database backend.

The JMS [31] 1.0.2b compliant middleware provider is configured to use persistent message queues for reliable message delivery and comes with an event-based model for message handling. The deployment of a message oriented middleware solution is crucial to the design of the framework discussed since it ensures reliable delivery (exactly once) even in the presence of network partitions and server crashes.

The combination of application server and message oriented middleware allows us to preserve transactional integrity by relying on the fact that both components negotiated the required global transaction information transparently. The deployment of message driven beans allows us to leverage the full potential provided by asynchronous message oriented middleware.

3.2 Design

We introduce an additional *Data Service Layer* that abstracts data access and mediates between business logic and database, handling dissemination and caching transparently. It is important to note that the Data Service Layer relies on standard persistence and database access technologies, namely container managed persistence and JDBC. This makes it possible to leverage the full potential of the Java 2 Enterprise Edition including transactional database access, security, resource pooling and concurrency. Furthermore, it enables integration of an adaptive data dissemination and caching layer into existing business applications that utilize data access mechanisms suggested by the J2EE.

The diagram showed in Figure 2 outlines the typical application programming model of the Java 2 Enterprise Application with the additional Data Service Layer that implements the adaptive data dissemination and caching logic. The same component is deployed at the backend server and each edge server; configuration of the functionality provided by the Data Service Layer relies solely on deployment descriptor elements. It should be noted that the pull logic is deactivated at the backend server.

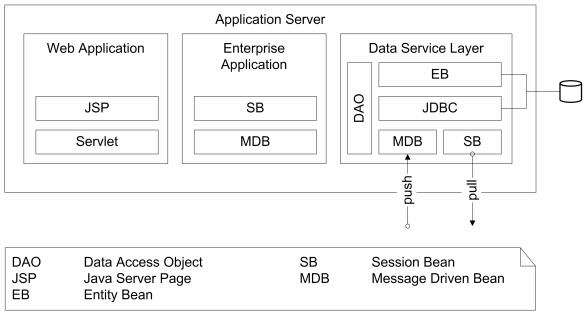

Figure 2 Application Model – Data Service Layer

The presented framework promotes loose coupling by introducing an abstraction of slightly restricted interfaces for data access. This abstraction hides implementation details of the Data Service Layer and streamlines the addition of business logic. The interfaces relieve software engineers from the burden of handling consistency within the business logic by adding reconciliation rules for generic database interfaces.

The Core J2EE Patterns [7] capture Sun Microsystems' current view of the preferred way to build enterprise applications for the Java 2 Enterprise Edition. Besides several core J2EE patterns introduced for convenience and flexibility, the framework defines the Data Service Layer in terms of Data Access Objects (DOA) [7] that encapsulate the conventional persistence code. The data dispatching and caching logic implemented in the Data Service Layer is completely shielded from the application logic. More sophisticated abstractions such as the Value Object Dispatcher Pattern [26] have been evaluated but we determined that the type safe nature of the Data Access Object pattern results into a better integration with the business logic. Furthermore, we believe that a production-ready framework would be integrated into an object-to-relational mapping tool that relies on some sort of code generation as described later in this paper.

The UML component diagram in Figure 3 shows the Data Service Layer. The diagram shows the deployment at an edge server as outlined above the pull logic is deactivated at the backend system. The *Data Dispatcher* is the central component of the framework responsible for accessing the local database through container managed persistence entity beans and JDBC. In addition the Data Dispatcher is responsible for coordinating the push and pull communication with the business participants. The *Decision Logic* component handles the dispatching and caching logic and encapsulates the expiration mechanism. Well defined extension points enable the integration of application logic into data dissemination and caching decisions. The *Common Services* component may be used for signing and encrypting data transferred among business partners and could be used to maintain usage statistics.

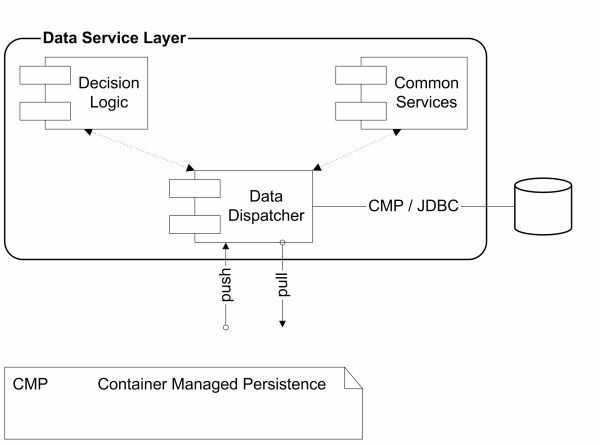

Figure 3 UML Component Diagram – Data Service Layer

The utilization of JMS [31] for proactive data dissemination fits the framework's requirements and was strengthened by a recent IBM case study [10]. A more detailed examination of the performance gains resulting from the deployment of asynchronous message based processing can be found in a recent research study [34]. The publish-and-subscribe paradigm of JMS topics allows to significantly reduce the communication overhead especially in systems with a large number of edge servers. It allows the backend system to asynchronously distribute notifications and data items to all edge servers. In contrast to synchronous data dispatching the backend system is not required to wait for data transmissions to every edge server to complete; instead it is possible to continue processing directly after the message left the system therefore increasing performance by several orders of magnitude.

The loose coupling resulting from the utilization of message oriented middleware allows adding new edge server nodes without changing or reconfiguring the framework deployed. We are currently investigating if asynchronous communication can also be applied to pull based information retrieval in case of cache misses. At the moment we think that direct remote method invocation is best suited and is currently being implemented in the prototype implementation. In order to implement edge server initiated data retrieval, we would need to transform synchronous cache lookups into asynchronous data requests. Implicit wait can be achieved with combinations of several entity beans and session beans [3] or probably with the adoption of the architectural Half-Sync/Half-Async pattern [9] to the J2EE platform.

In order to process asynchronous notification and proactively disseminated data items each edge server is equipped with a message driven bean that acts as a Service Activator [7]. On message retrieval the Service Activator delegates the notification or data item to the Data Service Layer. The Timer Service of the evolving J2EE 1.4 standard [36] will allow us to utilize a standardized scheduling system for data expiration. Currently we are utilizing a vendor specific timer service integrated into the application server in use.

3.3 Implementation

The implementation of the framework applied to the edge server architecture is being developed as flexible as possible to enable the integration into other application scenarios as outlined in the preceding sections.

We use the Abstract Factory strategy of the Data Access Object (DAO) pattern [7] to create backend system and edge server specific Data Access Objects, depending on the deployment node. In this way we can rely solely on information provided within the deployment descriptors to configure the abstract factory and to instantiate the correct set of related objects. Using the abstract factory at the backend system will create objects specific to notification and data dispatching, while objects created at the edge service will be responsible for handling cache misses.

It is evident that each DAO at the backend system and each DAO at the edge server is following a common algorithm to realize the frameworks' purpose. As with a regular DAO at the backend system, each object mediates between the business objects and the encapsulated database performing update, insert, delete, and lookup operations. In addition, for each relevant data access, a notification message is sent to the JMS topic and it is decided whether the data item affected by the data access is proactively pushed to interested edge servers. Just before the message is finally dispatched, we apply common services such as security and encryption to the message. At the edge server each DAO checks the local cache and upon cache miss sends a request directly to the backend system in order to retrieve the data item needed. Once again we will apply common services before issuing the pull request if necessary.

The DAOs within the framework follow a set of template methods [11] in order to provide a common algorithm and facilitate code reuse. Both abstract methods that determine required algorithm parts and hook functions that define optional extension points have been used. In this way the frameworks' behavior is localized in a common class and is easy to understand. Furthermore, it streamlines the addition of new DAOs that simply implement the variant parts of the predefined algorithm. Further aggregation and code reuse of common code parts of individual DAOs was achieved with the help of the Reflection API [22].

The dispatching logic at the backend system as well as the caching and expiration logic at the edge server is following the strategy pattern [11], which allows optimizing the overall performance of the framework by integrating new algorithms. The use of the strategy pattern enables changing the dispatching and caching logic at runtime to adapt to changing conditions. Furthermore, the algorithm's complexity and its specific data structures are not exposed to the clients using the algorithm resulting into a clean

interface. Therefore the integration of business logic into the dispatching, caching and expiration logic can be easily performed.

We are currently considering the use of the Decorator pattern [11] to add common services functionality to communications among business partners. This would enable us to transparently apply one or more value added services such as signing and encryption.

We believe that parts of the framework could be created with the help of code generation tools, which allow automatic creation of DAO objects for the data used by the enterprise application. In this way object-to-relation mapping tools could provide the frameworks' logic as an optional value added feature.

Therefore, allowing software developers to concentrate on the definition of dispatching, caching, and expiration logic required for each specific business application and facilitating the integration of the framework into existing enterprise applications.

If a conventional DAO layer is already present in an existing enterprise application, the code of the framework could be integrated by facilitating aspect oriented programming [4] methods or by tailoring attribute oriented programming tools such as XDoclet [41] to the frameworks needs.

4. RELATED AND FUTURE WORK

The presented framework is an approach to evaluate if adaptive and speculative data dissemination and caching is applicable for the interconnection of multiple business participants using Java 2 Enterprise Edition technologies. Several aspects require more detailed examination and the evaluation in different application domains and scenarios. Further research includes a detailed performance evaluation of the DAO layer extensions introduced by the framework in comparison to a regular DAO layer without data dispatching and caching functionality. Further evaluation of concrete algorithms for dispatching and caching as well as a detailed traffic monitoring in real world applications is scheduled to circumstantiate the benefits of the discussed framework in respect to network traffic and storage requirement reduction. It is inherent difficult to predict the consequences on scalability, availability and performance that result from the integration of the proposed framework into a general edge service architecture. Further research and theoretical evaluations are crucial to the generalization of the concepts presented.

As outlined above, addition of on-demand data broadcasting [19] upon edge server initiated data retrieval might result into an overall performance increase. The integration could be easily achieved however the implications in the discussed problem domain require further investigation.

We briefly announced the integration of workflow management techniques into the framework in order to achieve an optimized business processing collaboration. This allows an extension of data dissemination and caching functionality with support for sophisticated task delegation in edge service architectures. If we can find a methodology that allows us to exactly determine the data required by individual tasks we could significantly improve the overall performance of the framework. Further synergies can be exploited by utilizing cache aware query routing currently well known in the field of federated databases.

The persistent state service prototype [6] presents a very similar approach tailored to the CORBA platform. In contrast to the presented framework a complete unbundling of object caching and object-relational mapping is applied. This would require edge servers to keep cached data items in memory and therefore reduces the maximum amount of data that can be cached. Furthermore, the utilization of a relation database to store cached data items combined with a full index of available data items comes with numerous benefits. It enables the framework to access cached items exactly the same way it would access items at the backend system, namely through entity beans. This allows us to leverage the caching, resource pooling, and transactional benefits of the J2EE. In addition, standard query technologies such as JDBC and EJB QL (both find and select methods) can be performed on the cached data. The relational data cache further streamlines the integration into existing applications and the fusion with additional data that relies on a relational database system (the prevailing case in enterprise applications).

SpiritCache [35] from SpiritSoft is a commercial product that is based on the JCache (JSR 107) [37] initiative to provide a flexible and scalable caching framework [27] for the J2EE. It utilizes JMS for inter-cache communication and supports a number of value added services such as fault-tolerance and transactions required for critical enterprise solutions. Similar to the persistent state service prototype introduced in the previous paragraph the SpiritCache approach defines a clean separation between the business tier and the data backend. Our framework is tailored to distribute both business logic and business data. This allows us to leverage context specific knowledge of the backend system and individual edge servers to determine the optimal data dissemination and caching algorithm at runtime. Furthermore, it comes with the benefit of distributing business processing in addition to business data to multiple servers.

A detailed compilation with a huge spectrum of business to business interaction technologies and the issues involved can be found in a well structured publication [5]. The demonstration of an optimized data dissemination middleware can be found in [20], while the application of design patterns in the development of middleware solutions can be found in [8].

Another research project presented in [21] increases performance and availability of enterprise applications that utilize the edge service architecture by introducing slightly relaxed consistency for data replication. We argue, however, that complete data replication (at least for parts of the data used in typical e-commerce applications) is not feasible. In contrast, our approach combines a replicated data index with adaptive data dissemination and caching for data items. The resulting data storage reduction especially in application scenarios with a large number of edge servers could be significant. In addition, the deployment of the framework inside an application server allows us to leverage services such as resource pooling, caching, and transactional support that aren't available by simply relying on a Servlet engine. However, a lot of innovative ideas introduced for relaxed consistency replication can be utilized for business data propagation in edge sever architectures where the presented framework is not feasible.

5. CONCLUSION

We discussed the need for efficient data dissemination to support the increasing number of edge service architectures and business to business collaborations performed over wide area networks and the internet. Studies have shown that adaptive push and pull approaches can significantly reduce the communication overhead in

large scale distributed systems. Our research focuses on the seamless integration of adaptive dissemination and caching techniques into existing enterprise applications built on top of the Java 2 Enterprise Edition.

In this paper we introduced an architecture that combines a number of existing J2EE technologies to construct an additional layer that mediates between the business logic and the data storage. Reliable and efficient publish-and-subscribe based notification propagation and data dissemination is achieved by the deployment of a message oriented middleware product. Direct communication in case of cache misses relies on conventional remote method invocation.

The communication infrastructure provided by the framework allows both edge servers and the backend systems to initiate interactions. This enables us to incorporate the knowledge of all business participants in order to optimize the number of push and pull operations and the data storage requirements.

It is important to take into account that the efficiency of the presented architecture is influenced by the structure of the data and the way data is used for processing. The framework was designed for application scenarios that depend on timely notifications of data changes and exhibit access patterns that are typical of e-commerce applications and monitoring applications.

The increasing data storage demands combined with the ubiquity of widely distributed enterprise systems requires the integration of business logic and data characteristics into data dissemination decisions. We believe that the discussion of the presented framework may aid the integration of adaptive data dissemination techniques into the domain of distributed enterprise systems.

The integration of relaxed consistency and adaptive approaches into an application domain where absolute reliability and transactional processing is a key requirement comes with numerous challenges. The inherent complex nature of distributed enterprise applications makes the prediction of implications resulting from the addition of heuristic behavior into such systems extremely difficult. Further theoretical and practical research is required to define guidelines and to determine integration issues.

The main contribution of this paper is to provide a flexible architecture that allows the integration of adaptive techniques into distributed enterprise applications. Furthermore, we demonstrate that speculative proactive message oriented data dissemination and caching can provide an efficient communication infrastructure for edge service architectures without the inherent overhead of complete data replication.

6. REFERENCES

[1] A. Abecker, Ansgar Bernardi, K. Hinkelmann, O. Kühn, and M. Sintek; Context-Aware, Proactive Delivery of Task-Specific Information: The KnowMore Project; Information Systems Frontier; 2000; ISSN 1387-3326; Kluwer

[2] A. Bestavros; Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems; Proceedings of the 1996 International Conference on Data Engineering; 1996

[3] A. Hsiung; Asynchronous Load Processing, Asychronous Multiple Load Processes (Reducing roundtrips); 2002; http://www.theserverside.com/patterns/thread.jsp?thread_id=11004&article_count=11#39067; TheServerSide.com

[4] B. Burke and A. Brock; Aspect-Oriented Programming and JBoss; 2003, http://www.onjava.com/pub/a/onjava/2003/05/28/aop_jboss.html, O'Reilly ONJava.com

[5] B. Medjahed, B. Benatallah, A. Bouguettaya, A. H. H. Ngu, A. K. Elmagarmid; Business-to-business interactions: issues and enabling technologies; VLDB Volume 12 Number 1; 2003; ISSN 1066-8888; Springer-Verlag

[6] C. Liebig, M. Cilia, M. Betz and A. Buchmann; A publish/subscribe COBRA persistent state service prototype; IFIP/ACM International Conference on Distributed systems platforms; 2000; ISBN 3-540-67352-0; Springer-Verlag

[7] D. Alur, J. Crupi, and Dan Malks; Core J2EE Patterns: Best Practices and Design Strategies; 2001; ISBN 0-13-064884-1; Prentice Hall

[8] D. C. Schmidt and C. Cleeland; Applying Patterns to Develop Extensible and Maintainable ORB Middleware; Submitted to the IEEE Communications Magazine; 1998

[9] D. C. Schmidt and C. D. Cranor; Half-Sync/Half-Async: An architectural pattern for efficient and well-structured concurrent I/O; In Proceedings of the Second Annual Conference on the Pattern Languages of Programs; 1995

[10] D. Drasin; Using JMS technology as a data replication solution; developerWorks; 2002; http://www-106.ibm.com/developerworks/ibm/library/i-jms/; IBM Corporation

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides; Design Patterns; 1995; ISBN 0-201-63361-2; Addison-Wesley

[12] F. Casati and M.-C. Shan; Event-Based Interaction Management for Composite E-Services in eFlow; Information Systems Frontier; 2002; ISSN 1387-3326, Kluwer

[13] F. E. Bustamante, Patrick Widener, and Karsten Schwan; Scalable Directory Services Using Proactivity; Proceedings of the Proceedings of the IEEE/ACM SC2002 Conference; 2002; ISBN 0-7695-1524-X; IEEE Computer Society

[14] G. Shegalov, M. Gillmann, and G. Weikum; XML-enabled workflow management for e-service across heterogonous platforms; 2001; ISSN 1066-8888; Springer Verlag

[15] J. Farley, W. Crawford, and D. Flanagan; Java Enterprise in a Nutshell; 2002; ISBN 0-596-00152-5; O'Reilly & Associates

[16] J. Gwertzman and Margo Seltzer; The Case for Geographical Push caching; In Proceedings of the 1995 Workshop on Hot Operating Systems; 1995

[17] K. Geihs; Middleware Challenges Ahead; Computer Volume 34, Number 6; 2001; IEEE

[18] K. Gerwig; The push technology rage…so what's next?; netWorker Volume 1, Number 2; 1997; ISSN 1091-3556; ACM Press

[19] K. Kothandaraman; On-Demand Data Broadcasting; Thesis; 1998; Texas A&M University and Louisiana State University

[20] K. Stathatos, N. Roussopoulos, and J. S. Baras; Adaptive Data Broadcast in Hybrid Networks; Proceedings of the 23rd VLDB Conference; 1997

[21] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar; Application specific data replication for edge services;

Proceedings of the twelfth international conference on World Wide Web; 2003; ISBN 1-58113-680-3, ACM Press

[22] M. Campione, K. Walrath, and A. Huml; The Java Tutorial Continued: The Rest of the JDK; 1998; ISBN 0-201-48558-3, Addison-Wesley

[23] M. Franklin and S. Zdonik; A framework for scalable dissemination-based systems; Proceedings of the 1997 ACM SIGPLAN conference on Object-oriented programming systems, languages and applications; 1997; ISBN 0-89791-908-4; ACM Press

[24] M. Franklin and S. Zdonik; "Data in your face": push technology in perspective; Proceedings of the 1998 ACM SIGMOD international conference on Management of data; 1998; ISBN 0-89791-995-5; ACM Press

[25] M. Sayal, F. Casati, U. Dayal, M.-C. Shan; Integrating Workflow Management Systems with Business-to-Business Interaction Standards; Proceeding of the 18th International Conference on Data Engineering; 2002; IEEE

[26] N. Holbrook; Value Object Dispatcher Pattern (Revised); J2EE Patterns Repository 2003; http://www.theserverside.com/patterns/thread.jsp?thread_id=19020&article_count=16; TheServerSide.com

[27] N. Thomas; Building Scalable Web Applications / Web Services Using JCACHE; JMS and XML; 2002; SpiritSoft LTD

[28] Oracle Corporation; Oracle9iAS TopLink; Oracle9i Application Server; 2003; http://www.oracle.com/ip/deploy/ias/gs/index.html?oracle9iastoplink.html; Oracle Corporation

[29] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy; Adaptive push-pull: disseminating dynamic web data; Proceedings of the tenth international conference on World Wide Web; 2001; ISBN 1-58113-348-0; ACM Press

[30] Q. Chen and M. Hsu; Inter-Enterprise Collaborative Business Process Management; Proceeding of the 18th International Conference on Data Engineering; 2001; IEEE

[31] R. Monson-Haefel and D. A. Chappel; Java Message Service; 2001; ISBN 0-596-00068-5; O'Reilly & Associates

[32] S. Acharya, M. Franklin, and S. Zdonik; Balancing push and pull for data broadcast; Proceedings of the 1997 ACM SIGMOD international conference on Management of data; 1997; ISBN 0-89791-911-4; ACM Press

[33] S. Acharya; Broadcast Disks: Dissemination-based Data Management for Asymmetric Communication Environments; Technical Report CS-97-15 Department of Computer Science; 1997; Brown University

[34] S. K. A. Buchmann; Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services; Proceeding of the 28th International Conference on Very Large Data Bases; 2002

[35] SpiritSoft LTD; SpiritCache; 2002; http://www.spirit-soft.com/products/cache/introducing.shtml; SpiritSoft LTD

[36] Sun Microsystems, Inc.; Java 2 Platform, Enterprise Edition Version 1.4 Beta 2 Release; 2003; http://java.sun.com/j2ee/1.4/download-beta2.html; Sun Microsystems, Inc.

[37] Sun Microsystems, Inc.; Java Specification Request 107 - JCACHE - Java Temporary Caching API; 2001; http://www.jcp.org/en/jsr/detail?id=107; Sun Microsystems, Inc.

[38] Tangosol Inc.; Tangosol Coherence; 2003; http://www.tangosol.com/coherence.jsp; Tangosol Inc.

[39] Thought Inc.; CocoBase; 2003; http://www.thoughtinc.com/cber_index.html; Thought Inc.

[40] W. D. Smith; TPC-W:Benchmarking An Ecommerce Solution; 2001; Intel Corporation

[41] XDoclet Team; XDoclet Attribute Oriented Programming; 2003; http://xdoclet.sourceforge.net/; XDoclet Team