IBM® WebSphere®

Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale

- Introduction to caching
- Concepts and considerations
- Real-world scenarios

Jan Bajerski
Ulas Cubuk
Fernando Ewald
Susan Hanson
Kris (Krzysztof) Kobylinski
Jun Guo Ku
Anton Litvinov
Nikolay Marin
Dmitry Savustjan

Redbooks

ibm.com/redbooks

International Technical Support Organization

Enterprise Caching Solutions using WebSphere DataPower SOA Appliances and WebSphere eXtreme Scale

April 2013

SG24-8043-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (April 2013)

This edition applies to:
- WebSphere® eXtreme Scale, Version 8, Release 5, Modification 0
- WebSphere Message Broker, Version 8, Release 0, Modification 0, Fix Pack 1
- WebSphere SOA DataPower Appliance XC10
- WebSphere SOA DataPower Appliance XI52
- IBM Worklight, Version 5.0

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

Preface

In the dynamic business environment of today, Information Technology (IT) organizations face challenges around scalability and performance. This IBM® Redbooks® publication is targeted for IT architects, IT personnel, and developers who are looking to integrate caching technologies, specifically *elastic caching*, into their business environment to enhance scalability and performance. Although it is helpful to know caching technologies, an introduction to caching technologies in general is included. In addition, technical details are provided about implementing caching by using several IBM products.

The IBM WebSphere® eXtreme Scale product provides several functions to enhance application performance and scalability. It provides distributed object caching functionality, which is essential for elastic scalability and next-generation cloud environments. It helps applications process massive volumes of transactions with extreme efficiency and linear scalability. By using the scalable in-memory data grid, enterprises can benefit from a powerful, high-performance elastic cache.

The IBM WebSphere DataPower® XC10 Appliance enables your business-critical applications to scale cost effectively with consistent performance by using elastic caching in a purpose-built, easy-to-use appliance.

This publication explains the benefits of using various caching techniques in your enterprise, specifically involving the use of IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 Appliance. Three real-world scenarios are described that use these enterprise caching technologies to solve issues that face the businesses of today.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the IBM Software Group Technical Exploration Center in Moscow, Russian Federation.

Jan Bajerski is a WebSphere Connectivity IT Specialist in Software Group Community of Practice in the CEE region and has worked in the IT industry for 11 years. Previously, he worked in IBM Software Services for WebSphere in Poland, supporting customers in implementing solutions using WebSphere Application Server, WebSphere MQ, and WebSphere Message Broker. He has a BSc degree from Warsaw University of Technology (Poland).

Ulas Cubuk is a Certified IT Specialist for WebSphere in IBM Software Group Turkey and has worked in the IT industry for 12 years. Over the last three years, she has worked with many clients on service-oriented architecture (SOA) projects, in particular, the business process management (BPM) and connectivity aspects to help clients in their particular business needs. Her current areas of expertise include BPM products and WebSphere DataPower and WebSphere ESB. In her previous roles in IBM, she worked with clients extensively on IBM Rational® IBM System z® development and Security and Compliance Management solutions. Before joining IBM, she focused on software quality assurance, application lifecycle management, software configuration management and she contributed to CMMI SCAMPI appraisals as an auditor. She has a BSc degree in Environmental Engineering and an MSc degree in Geographical Information Technologies from Middle East Technical University.

Fernando Ewald has 12 years experience in IT solutions. He joined IBM in 2009 as an IT Specialist for IGA Canada - Common Development and Test (CDT) supporting internal IBM Accounts as a member of the Innovation and Technical Leadership Team. Fernando's area of focus is Middleware Support including WebSphere DataPower Appliances, Information Server IBM DataStage®, reverse proxy, and other products. Before joining IBM, Fernando worked with a Sugar and Alcohol Company, creating high availability solutions for the industry and office. He also worked as a Teacher at Universidade de Franca - Brazil, where he taught Computer Science and System of Information courses, and Internet and Network Computer courses.

Susan Hanson is a member of the WebSphere Application Server foundation development team. She has 22 years of experience in developing and delivering IBM software products across the WebSphere and IBM Tivoli® brands. She holds a Bachelor degree in Computer Science from East Carolina University and a Master degree in Computer Information Systems from The University of Phoenix. She is based in Research Triangle Park, North Carolina and is temporarily working and residing in Shanghai, China.

Kris (Krzysztof) Kobylinski, an Advisory Software Engineer with IBM Canada, graduated from University of Manitoba with a Master degree in Computer Engineering and from Technical University of Wroclaw with a Master degree in Electronics. Currently, he is a member of the WebSphere eXtreme Scale development team. Before joining his current team, Kris participated in the development of various software products with the main focus on distributed execution tracing, logging and debugging, enterprise systems integration, data warehousing, and UI presentation of complex and dynamic data models. His interests include systems architecture, requirements acquisition, analysis and modeling, software systems performance, behavior patterns, human machine interaction, and autonomic computing. Kris is the author and co-author of a number of patents.

Jun Guo Ku is a Software Engineer from the IBM China Software Development Lab in Beijing, People's Republic of China. He has been working in the WebSphere Application Server foundation team for over six years. His areas of expertise include WebSphere Application Server on IBM z/OS®, Java programming, and web development. He holds a Bachelor degree in Mathematics and a Master degree in Computer Science from Fudan University, China.

Anton Litvinov is a WebSphere Connectivity IT Specialist in the IBM Software Group Community of Practice in Central and Eastern Europe. He has over 11 years of experience in the IT Industry, out of which five years were spent working with IBM Software Group in the connectivity space focusing on DataPower appliances. He has a Bachelor degree in Computer Science from Moscow State University of Instrument Engineering and Informatics.

Nikolay Marin has six years of experience with IBM and more than 10 years of experience in the IT industry. He is an Open Group Certified Distinguished (Chief/Lead) IT Architect. Currently, Nikolay works closely with clients on projects in the telecommunications industry and cloud computing solutions across the Central and Eastern Europe region. Before his current role, he focused on WebSphere solution sizing and Performance Management as an IT Specialist with IBM. Nikolay holds a Ph.D. degree in Civil Engineering and an MBA degree in Management.

Dmitry Savustjan has a great interest of the mobile field. He shares his passion and work. Currently, he is the IBM Worklight Technical Lead for the Central Eastern Europe region with IBM. He holds a Master degree in Computer Science from Moscow Engineering Physics Institute Technical University. His areas of expertise include mobile development, Java development, application design, and infrastructure design.

Thanks to the following people for their contributions to this project:

Tamikia Barrow, Deana Coble, Linda Robinson, Carla Sadtler, Stephen Smith, Margaret Ticknor, and Debbie Willmschen
International Technical Support Organization, Raleigh Center

Anna Shugol and Konstantin Yurov
IBM Russia, Moscow

Yakura Coffee, Tom Gissel, Todd Kaplinger, Charles Le Vay, and Brian Martin
IBM US, Raleigh

Anton Aleksandrov
IBM Israel, Haifa

Bin Li and Xiao Xia Qiu
IBM China, Shanghai

James Hart and Matt Lucas
IBM UK, Hursley

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction to IBM Enterprise Caching Technologies

This IBM Redbooks publication describes the concept of caching and its value for your business. It also describes the IBM products, tools, and techniques that you can use to build a caching solution for your enterprise or for a client.

The book does not go into detail about all options or every available topology. Instead, it focuses primarily on a caching tier that can be part of any topology.

This chapter provides an introduction to caching and caching technologies and provides a basis for the remaining chapters in the book. It includes the following topics:

1.1 Overview of caching

Caching is one of the general approaches that are used to mitigate the risks of poor performance or limited scalability. In the same way that humans and animals store frequently used resources close to where they are consumed, systems can also benefit from keeping their resources close by.

Imagine that you cook pizza at home. To make a good pizza you need good cheese and fresh tomatoes, so a good cook goes to the market to buy fresh ingredients each time.

Now, imagine that you own a pizzeria and cook many pizzas every day. You cannot go to the market for cheese every time someone orders a pizza. Instead, you buy cheese once a week and put it in a refrigerator (a cache) where it is readily available.

Fresh tomatoes are also needed. This component is more complicated because ripe tomatoes cannot be stored for long periods of time. To ensure fresh tomatoes, an assistant goes to the market each morning to buy tomatoes. This process ensures that you always have good cheese and fresh tomatoes in a cache (the refrigerator).

When your business expands, you start to serve pasta in addition to pizza. Unfortunately, your suppliers can deliver pasta only overnight, when you are asleep. So they agree to leave the pasta you order in a small storage area (another cache), where your assistant can pick it up each morning.

Meanwhile, a water service delivers bottled water to your pizzeria. They can deliver it at any time, but they charge a fixed price, independent of volume, for each delivery. Because of the fixed cost for deliveries, you decide it is better to store a large volume of water, enough for one week, which allows the supplier to be called less frequently to keep costs down.

These concepts are similar in enterprise IT systems, where caching techniques are now broadly used by application developers and engineers.

The idea in an IT environment is simple: store data in-memory to reduce the need for frequent communication with databases or services over a network. Things get more complicated when the data needs not only to be read but also updated (written), but there are ways to manage this complicating factor. You can also have situations where external data providers charge per request. Depending on the volatility of the information, caching the data from the external provider can save money by reducing the frequency of those requests, reducing the costs that are incurred.

1.2 Benefits of caching to an enterprise

Service industries such as retail, banking, travel, and insurance are already well established in cyberspace and are reaping the benefits of providing the *buy at your own convenience* business model to their customers. This business model is widely used and a number of IT solutions support it well.

However, the introduction of mobile computing devices, such as tablets and smartphones, redefined performance requirements for enterprise IT systems. The new *connected all the time* environment introduced new behavior patterns where the use of social media is an everyday activity for a large part of society. Together, these new realities are calling for new solutions.

Existing solutions cannot support the data volume overload at a performance level that is satisfactory to users. If it takes too much time to refresh a page or complete a transaction, business can be lost and, in some cases, the client will not come back. More data must be provided in less time. Data availability and quick access times become critical. In this environment, data caching is a powerful mechanism to rejuvenate existing enterprise IT solutions to better match these new requirements.

Growing enterprises are trying to use their existing infrastructure at its highest capacity. In a cost-effective manner, these organizations must ensure that crucial applications can meet performance requirements and scale to meet increasing demand.

A modern enterprise can benefit from introducing caching into its IT systems in five important areas:

- Cost saving
- Scalability
- Availability
- Failover
- Flexibility

1.2.1 Cost saving

Typically, database workloads are placed on expensive hardware. HTTP servers and application servers are able to run efficiently on multiple, lower-range machines. However, analytical and transactional databases usually require a smaller number of mid-range and top-range nodes to work efficiently. This process makes scaling these databases expensive.

Adding caching into this environment lets you use multiple industry-standard machines to store and access large volumes of frequently used and updated data.

In some cases, external data providers charge for each access of the data from their systems. If this information does not change with each transaction, caching of this data can reduce the charges that are incurred by an enterprise.

1.2.2 Scalability

Many enterprises today face exponential growth in the volume of transactions and exploding numbers of mobile devices accessing their services. Companies are looking for cost-effective ways to enable high transactional workloads and large data volumes for applications that span multiple servers.

In addition, cloud computing and the growing demand for mobile data access present these new challenges:

- Significant growth in transaction rates and data volumes
- Growth of processor load and memory consumption on back-end systems
- Constant need to scale in and scale out front-end systems

In some cases, data sets become so large and complex that it becomes increasingly difficult to process the data by using traditional database management tools. These data sets, called *big data*, require different technologies to process the data. Although these big data solutions are often centered around new database technologies that might not be primarily based on tables, their architecture and application environments increase the demand for scalability of relational databases.

Implementing caching can provide a rapid and elegant way to eliminate the scalability demands for existing databases, without the need to apply modifications to them. Frequently accessed data from a database can be stored in the cache and the applications can retrieve the data from the cache instead of going directly to the database. By using the cache, the applications have faster access to the data. The applications have the added benefit of reducing the number of times the database is accessed.

1.2.3 Availability

Companies often must grapple with inconsistent application response times, yet system response time, availability, and reliability are critical in providing high-quality IT and business services. One of the technologies available to help in this area is *session persistence*, which is an efficient way to share the servlet state information across multiple application servers that run the same application. By storing this state information in an external data store, such as a database, a request by a user can be handled by different application servers that are based on availability. Add a cache in front of the database, and you increase the availability of the data without adversely impacting your database management system. Session persistence, in general, can dramatically improve the performance of application servers and portals of large and geographically distributed service providers.

Many applications today are *composite applications*, aggregating data from various sources that can be local, federated, or cloud-based. Availability and performance in each of the application tiers directly impacts total availability. A bottleneck in any of the application tiers can make the applications unavailable or increase response times. Caching the data, where the same data is replicated in multiple places and potentially in multiple locations, is the answer. When you have a single data store, if that data store is heavily burdened or becomes unavailable, the application no longer has the necessary data to continue. By having a distributed cache, if one instance of the data is unavailable, the data is still accessible from the other instances.

1.2.4 Failover

To keep data consistent in distributed systems, replication of transactions and data change notifications must be organized. Luckily, several approaches are available to ensure consistent data across distributed systems.

Distributed caching supports different failover scenarios. With a cluster of WebSphere eXtreme Scale containers, a cluster node can fail without the loss of data because each data object is copied to two or more nodes. These WebSphere eXtreme Scale containers can be stand-alone or embedded in a Java process such as a WebSphere Application Server or WebSphere Application Server V8.5 Liberty Profile server. The data is redistributed across available caching nodes after failover detection. Another option in the case of failover is memory-to-memory replication of state across multiple systems in various data centers.

Caching can also virtually eliminate an enterprise's need for costly databases to handle HTTP session persistence, and can be used for multi-data center HTTP session failover. By using an elastic cache for HTTP session persistence, you can replace other session replication mechanisms such as memory-to-memory replication. By offloading the HTTP session cache to an elastic cache, you can reduce the memory requirements of your application server Java virtual machine (JVM). With this step, you can enhance memory usage and tune your application server environment.

1.2.5 Flexibility

A standard three-tier application (with separate modules for presentation, logic, and data) typically presents the following technical challenges:

- Increase in user workloads
- Need to reduce a system response time
- High database load, which causes slow data access
- Vertical and horizontal scalability of infrastructure to handle growing data volume

To meet these challenges, fast and flexible data access in the enterprise architecture becomes critical. For instance, acquiring a new company might slow down the order management process because one of its web-services is now located far away and responds slowly to requests. Caching can enhance integration and enable implementation of service-oriented architecture (SOA) by improving request response time and increasing service throughput. In addition, memory limitations of a Java virtual machine (2 GB on 32-bit operational systems, for instance) can be avoided by using a side cache.

1.3 Common caching patterns and scenarios

An efficient approach to caching in enterprise applications today is *distributed caching*, which allows the cache to span multiple servers. Distributed caching involves storing the cached data in a special logical tier, separate from your application tier. This configuration offers two main advantages:

- Ability to offload storage of the cached data and dedicate the resources to the application tier
- Ability to maximize scalability, because growing demand does not always require allocating more expensive hardware resources to the application tier (most of the workload is placed on the less expensive caching hardware platform)

Distributed caching helps companies lower the total cost of ownership (TCO) of their IT systems and enables high-performance computing, clouds, mobile applications, and extreme transaction processing.

The most advanced caching solution approach is the use of an *elastic cache*. This type of cache allows on demand scale-out and scale-in, and also provides self-healing by automatically recovering from failures.

This book focuses on the most common patterns and scenarios for using caching solutions in an enterprise infrastructure.

1.3.1 Caching patterns

The following caching patterns are the ones used today by most enterprises:

- Read-only

  This type of cache is used for static data to improve performance and scalability. It provides fast access to data that changes only occasionally. The data is owned by the back-end system and the data that is stored in the cache cannot be modified by the application. The data in a read-only cache might be loaded from a back-end database or from files during startup.

- Read/write

  This type of cache is used for volatile data. When applications can read, add, modify, or delete data in the cache, changes must be propagated from the cache to the back-end database. In this pattern, the cache has the master role and changes are written to the cache first, and then from the cache to the back-end database. If conflicts are detected, they are resolved by using predefined methods.

- Read-through

  This type of cache is similar to read/write cache, but the back-end database plays the master role. If data is not yet in cache, it is first added to the cache from the database.

- Write-through

  With this approach, changes made by an application are written to the cache and to the back-end database synchronously. This process might slow down the system on a write operation, but it is a simple method that ensures consistency of the data.

- Write-behind

  This type of cache services all data reads and updates, but unlike a write-through cache, updates are not immediately updated in the database. Instead, updates occur in real-time only in the cache. The cache tracks the updates that are made and periodically writes the current set of updates to the database. The database is updated asynchronously by a special loader. If the same record is updated multiple times within the buffering period, only the last update is written to the database. This technique can significantly improve performance in scenarios where values change frequently but all users of the database data must be aware that the cache might contain updates that are not yet made to the database.

All of these patterns are used for various purposes and can be combined.

It is important to mention that the cached data can be stored locally, remotely, or both. A cache can also be efficiently used for session management in multiple data centers.

1.3.2 Caching scenarios

The following list provides the three most popular solution scenarios for caching in an enterprise application infrastructure:

- Side cache for applications
- Side cache for enterprise service bus (ESB)
- Cache as an integration point

There are more caching scenarios used today, but they are not covered in this publication. In addition, HTTP session caching and geographically distributed caching are not covered in detail.

Side cache for applications

Use of a side cache for an application, as shown in Figure 1-1 on page 7, is the classic cache usage scenario. An application reads data that is stored in a *backing store*, which can be on a disk, in a database, or in some other back-end data store. The application first attempts to get the data from the side cache. If it finds it there, it uses that data and the application saves the time that it would take to read the data from the back-end data store. If the data does not exist in the cache, the application reads the data from the back-end data store and stores a copy of it in the cache. The next time that the application needs the same data, it follows the same process, but this time, it finds the data in the cache.

Figure 1-1 shows the use of a side cache for applications.

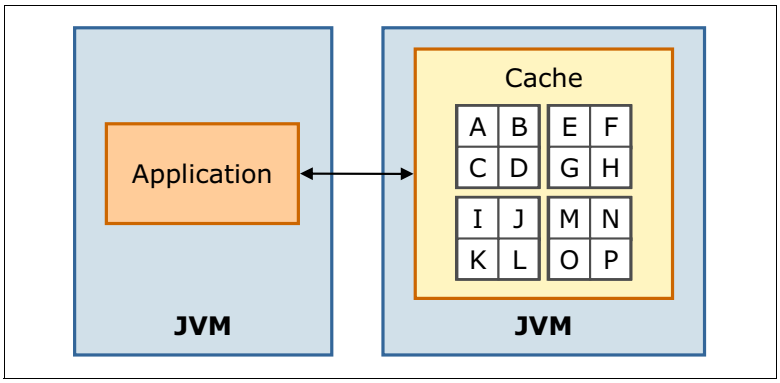

Figure 1-1 Side cache for applications

The side cache can also be preinstalled from the back-end data store when the application starts. This way, all of the data is already available in the cache when the application needs it. In addition, data that is generated by application computations can be stored in the cache for later use.

Side caches offer three significant benefits for cache hits:

- Reduced response time: Cache hits give you reads at cache speeds, not back-end speed. Response time and throughput can increase dramatically. Cache size and cache hit ratio (the ratio of cache hits to cache misses) are key factors.
- Reduced load on the back-end data store: Every request that is satisfied from the cache by a cache hit is one less request that is made to the back-end data store. Cache hits, therefore, eliminate redundant processing and reduce the load on the back-end data store.
- Moving stored data outside of the application JVM (and potentially outside of application machine): This process almost eliminates a well-known memory limitation of the Java virtual machine (2 GB on 32-bit operational systems, for instance).

The technical details of the side cache for applications scenario are covered in Chapter 3, “Enterprise caching in a mobile environment” on page 23.

Side cache for enterprise service bus

A side cache offers an easy approach to integrating caching with an ESB, as shown in Figure 1-2 on page 8, which is a critical component of any SOA. The ESB connects and integrates applications, services, and business process flows at the messaging layer. It does protocol mediation, message transformation, routing, and process choreography, and provides quality of service in terms of security, reliable message delivery, and transaction management.

Figure 1-2 shows the use of a side cache for an ESB.

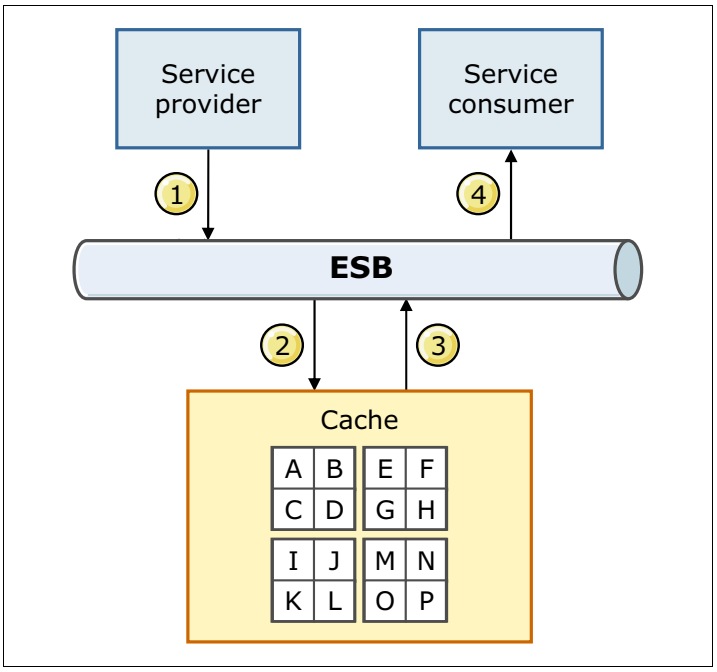

Figure 1-2 Side cache for an ESB

In an SOA application infrastructure, requests pass through the ESB before they are sent to the application. Therefore, if you can retrieve the result of an application request from the side cache, you can reduce the application processing and processing latency for that request. This results in a significant decrease in response time and reduced back-end application processing. For elastic caching solutions, the side cache operation is added to the ESB flow. Therefore, you do not have to change the application code. The ESB side cache is a common use case for a simple data grid.

From a technical standpoint, the addition of a side cache to an ESB can help in several ways:

- Significantly reduces the load on the back-end system by avoiding redundant requests
- Helps to avoid multiple costly redundant requests
- Facilitates more real work to be performed
- Improves overall response time
- Reduces the need to scale hardware to increase processing capacity because the back-end system no longer has to handle redundant requests
- Facilitates response times from elastic cache that can be measured in milliseconds

From a business standpoint, the addition of a side cache to an ESB provides two important benefits:

- Saves money by reducing the number of responses at charge (if your company is charged per request, but the data is not updated frequently by the provider)
- Integrates low performance systems into the data process by caching response of these systems and updating the cache asynchronously.

An example of this scenario is provided in Chapter 4, “Enterprise caching in an ESB environment” on page 45.

Cache as an integration point

Using a cache as an integration point, as shown in Figure 1-3, represents advanced grid usage and exploitation.

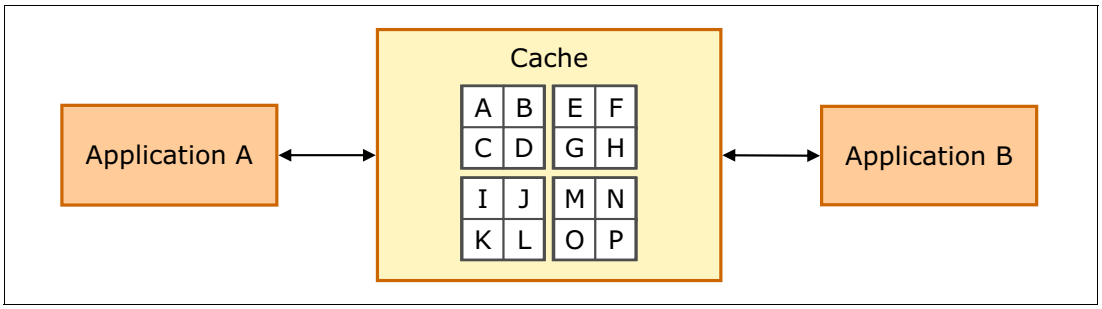

Figure 1-3 Cache as an integration point

There are two key ideas in this scenario:

- Moving processing logic to the data
- Using the MapReduce programming model

In a grid, the data is partitioned across grid servers. When an item is added to the grid, it can be processed by the grid server that stores it, thus moving processing logic to the data.

The MapReduce programming model is used for processing large data sets. Using this programming model allows processing to be distributed among a subset of the storing partitions in parallel, with the results consolidated at the client.

Performance improvements can be gained by computing at the data location. If the computation deals with large amounts of data, it is much easier to send the computational logic to the data and then process the data locally. If not, you must send large amounts of data to the computing location.

Solutions of this type often fall into the category of Extreme Transaction Processing (XTP). This solution is an application style that is aimed at supporting the design, development, deployment, management, and maintenance of distributed transaction processing applications. This solution is for applications with demanding non-functional requirements of performance (500 transactions/second and 10,000 concurrent accesses or more). Other requirements include scalability, availability, security, manageability, and dependability requirements.

Implementation of this scenario requires programmatic changes to your application. An example of this scenario is covered in Chapter 5, “Enterprise caching in an extreme transaction processing environment” on page 83.

Inline cache

In an inline cache scenario, which is shown in Figure 1-4 on page 10, a loader is used to move data between the cache and the ultimate backing store. With a write-through caching pattern, the transaction does not complete until the data is written to the backing store. With a write-behind caching pattern, the transaction completes when the data is written to the cache, and the cache asynchronously replicates the data to the backing store.

Often for multiple changes in a batch, only the last change must be written. Multiple changes to multiple fields of the same record are consolidated into a single write to the record. Writes are batched, and writing to a database in batches is more efficient than many individual transactions. Finally, the cache serves as a shock absorber in front of the database. The

cache accumulates the changes so even if the database goes down, perhaps because of lost connectivity, the application that uses the cache is not affected.

Figure 1-4 shows an example of an inline cache.

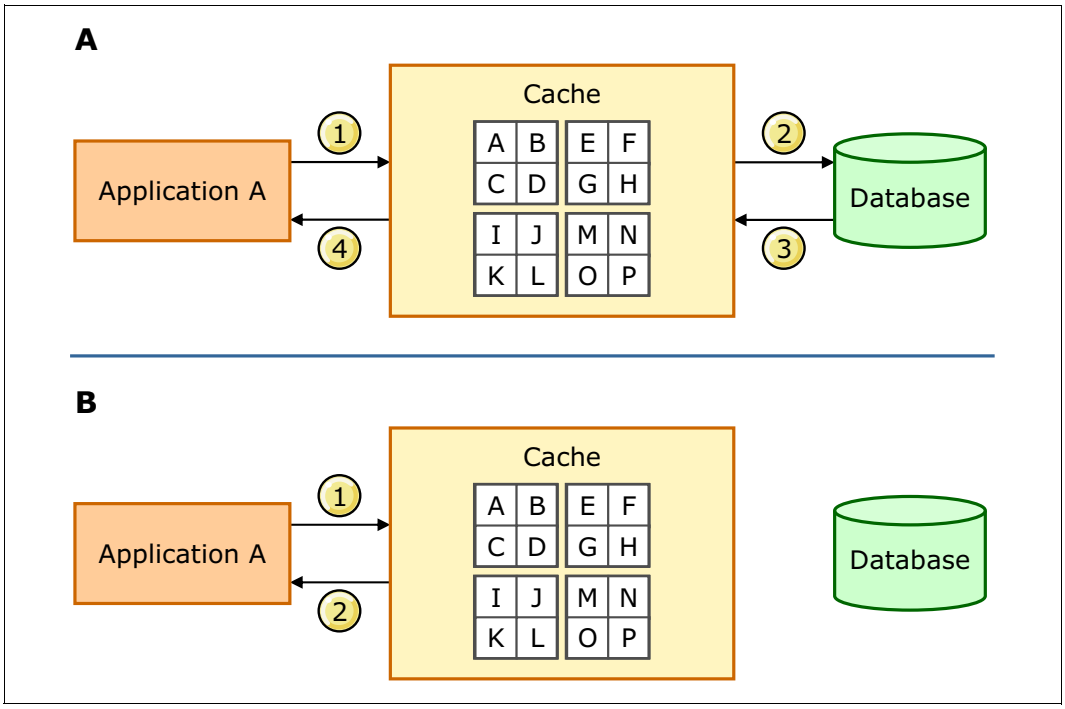

Figure 1-4 Inline cache

In Figure 1-4, scenario A (top) shows the flow of the request and data when the requested data is not yet in the cache. The application requests the data from the cache (1). The cache server determines that the data is not present and goes to the database (2) to get the data. The data is returned to the cache server (3), stored in the cache, and then provided to the application (4). In scenario B (bottom), the application requests the data as before (1) but this time, the data is in the cache and so the cache server just returns the cached value without going to the database (2).

Benefits of using inline cache include better throughput and more efficient use of the database, insulation from database load and failure, better response time, and reading at cache speed. With a write-behind caching pattern, the writes are done at cache speed as well. Although the concept of an inline cache is introduced here, the inline caching scenario is not covered as part of this book.

1.4 Overview of IBM WebSphere eXtreme Scale

The *IBM WebSphere eXtreme Scale* product provides an elastic, scalable, in-memory data grid. The product dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers. Applications can access data from the grid faster than from a traditional database. Adding IBM WebSphere eXtreme Scale to an IT environment helps improve performance by providing the following benefits:

- Eliminating disk storage as a performance bottleneck

- Reducing processor usage and latency that is associated with exchange of invalidation-related information between physically separate cache instances (for example, nodes within a cluster communicating changes in the cache among themselves)
- Reducing the time to reach *warm* state (when the cache is populated with data) after performing full site restart, partial site restart, and complete or partial cache invalidation

IBM WebSphere eXtreme Scale provides an extensible framework to simplify the caching of data that is used by an application. It can be used to build a highly scalable, fault tolerant data grid with nearly unlimited horizontal scaling capabilities. It allows memory capacity to be added (scale-out) or removed (scale-in) while the grid is running without requiring a restart. This *elastic scalability* is possible by using distributed object caching.

IBM WebSphere eXtreme Scale creates infrastructure with the ability to deal with extreme levels of data processing and performance. When the data and resulting transactions experience incremental or exponential growth, the business performance does not suffer. The performance does not suffer because the grid is easily extended by adding more capacity in the form of JVMs and hardware.

1.5 Overview of IBM WebSphere DataPower XC10 Appliance

The *IBM WebSphere DataPower XC10 Appliance*, shown in Figure 1-5, provides some of the same benefits of the IBM WebSphere eXtreme Scale product in an easy-to deploy appliance. Each XC10 appliance provides a large, 240 GB cache and a simplified management interface for creating and monitoring data grids, and administering and monitoring the appliance itself. Appliances can be grouped into a highly available *collective*, providing failover capability and increasing overall cache capacity and throughput.

Figure 1-5 WebSphere DataPower XC10 Appliance

The WebSphere DataPower XC10 Appliance enables accelerated time-to-value by reducing the time that is needed for installation, setup, and configuration through immediate, *drop-in* use. The appliance can benefit simple side cache scenarios, HTTP session replication, and WebSphere Application Server dynamic cache service.

The XC10 Appliance ensures high availability of data for mission-critical applications, scales with simplicity, and delivers high performance and consistent response times.

A major benefit to the XC10 Appliance is simplicity and the drop-in approach of an appliance. There is no programming that is involved and no changes necessary to applications. You can implement high availability by using at least two appliances. It takes only a short time to deploy after it arrives at your data center because the software is preinstalled.

The WebSphere DataPower XC10 Appliance also provides drop-in security. Because the device is physically closed, access is limited. Management is done through a single console, and both shell and unsecured remote access are disabled.

To provide scalability and high availability, appliances can be grouped into a *collective*. These collectives can be incorporated into different topologies, which are based on the needs of your enterprise:

- Line topology
- Ring topology
- Spoke and hub topology
- Tree topology

A simple *line topology* is just what it sounds like: a set of collectives that is linked together in a line, similar to the links of a chain. Changes are propagated from one collective to the other only by the collective it is directly linked to. Therefore, changes in the first collective are sent to the second in the chain, then from the second to the third, and so on. Changes from the first are never directly sent to the third. However, if a collective is down, changes are not propagated to collectives further in the chain until the collective that is down is restarted.

The *ring topology* is more resilient than the line topology. Think of the line topology and then connect the first and last collectives in the line to form a ring. Now, when a collective is down, the changes to the collectives further down in the chain can still be propagated from the collective on the other side, flowing the opposite direction through the ring.

With a *spoke and hub topology*, there is a central hub collective and all other collectives are connected to this hub. Changes are distributed to and from the hub to all of the other collectives. If the hub fails, changes are not distributed until the hub restarts.

In topologies where there are many collectives, a *tree topology* is useful. In a tree topology, you have a top collective and then levels underneath it, like a child-parent relationship. The top collective communicates to its child collectives, which in turn have child collectives that they communicate with.

The IBM WebSphere DataPower XC10 Appliance has a simplified management and administration capability, with an administration and monitoring console that enables efficient setup, configuration, and management of the appliance and transaction loads within your data center.

1.6 WebSphere eXtreme Scale versus DataPower XC10 Appliance

Both the IBM WebSphere DataPower XC10 Appliance and WebSphere eXtreme Scale software address scalability issues by introducing elastic data grid capability into enterprise solutions. The data grids hold cached data in non-SQL structures. Both enable scalability of transaction volumes with minimal changes to applications or architecture, and they reduce the number of read and write operations to databases. These benefits result in less time that is required for resource-intensive calls. However, the solutions are intentionally positioned differently.

The focus of the IBM WebSphere DataPower XC10 Appliance is on simple, data-oriented caching scenarios, where clients want to improve performance and scale their applications in a quick and cost-effective manner. Most data-oriented caching scenarios require few, if any, application code changes. In fact, using the XC10 for WebSphere Application Server session management or for extending DynaCache requires no application code changes. The XC10 provides a large cache in a high density, low footprint environment that saves time, money,

and rack space. Fault tolerance is built in, lowering the risk of data loss and providing continuous availability. The appliance is best suited for the following data-oriented solutions:

- HTTP session management
- Extension for DynaCache
- Simple grid or ESB side cache
- Worldwide cache, or a multi-data center cache

All of these scenarios can also be implemented by using WebSphere eXtreme Scale software, but it can also be used in a broader set of solutions. In addition to these scenarios, WebSphere eXtreme Scale software also fits into the following application-oriented solutions:

- Database buffering
- Business event processing
- Petabyte analytics
- In-memory online transaction processing (OLTP)
- In-memory SOA

The appliance is able to lower the total cost of ownership (TCO) of a caching solution because of its simplicity and drop-in approach. However, it is limited to a subset of the solutions available by using IBM WebSphere eXtreme Scale. The individual needs and requirements of the business help to decide which caching solution is best for you.

2

Concepts and considerations

The basic concepts of caching by using IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 Appliance are introduced. These concepts are important to understand before you proceed into the different example scenarios and solutions. Various factors to consider when you determine how to best design a caching infrastructure for your environment are also covered. The following topics are described:

2.1 Basic concepts

To understand the logical structure of a cache by using IBM WebSphere eXtreme Scale or the IBM WebSphere DataPower XC10 Appliance, you must have an understanding of the servers in the underlying caching architecture and how the data within the cache are structured by using maps, map sets, and object grids.

2.1.1 Container server

The *container servers* hold the actual application data for the cache and each runs in its own Java virtual machine (JVM). A local cache can have just one container server. A distributed cache has more than one container server, and those container servers are remote to each other.

2.1.2 Catalog server

Each cache should have at least one *catalog server* and the catalog server provides administration services to its domain. All container servers that register with the same catalog server define a *catalog server domain*.

The two main services that are provided by a catalog server are *location service* and *placement service*. The location service tracks where each object is located within the cache. Therefore, when a request to get an object is received by the cache, the catalog server knows where this particular object is located. The placement service places individual shards in such a manner that, under no circumstances, a primary shard and any of its replicas are in the same container, given that there is more than one container in the cache.

2.1.3 Maps

All data that is held in a cache is organized into *maps*. Each map holds objects of the same type. In essence, a map functions like a hash map where the data is stored as key-value pairs. The key is used to access the value that is associated with that key. Each key can be either a simple data type or a Java object and each value is a Java object. Maps allow data manipulations by supporting the basic operations of insert, get, update, and delete.

2.1.4 Map sets

Map sets are used to group one or more maps that have the same partitioning and replication policy. Each map can be distributed across multiple partitions. Map sets make sense only for distributed caches and are not used for cache instances that are local or embedded into the application JVM.

2.1.5 Object grids

An *object grid* is the highest entity in the logical structure of a cache. The object grid is sometimes referred to as a *data cache*, *data grid*, or just a *grid*. Each object grid can have multiple map sets. There can be multiple object grids in a single catalog server domain.

2.1.6 Partitions

Data within a map is generally broken into parts, called *partitions*. Partitions host a subset of the data in the grid. WebSphere eXtreme Scale automatically places multiple partitions in a single container and spreads out the partitions as more containers become available. It is important to choose the number of partitions carefully before final deployment because the number of partitions cannot be changed dynamically.

2.1.7 Shards

Shards are instances of partitions and can either be a primary shard or a secondary (or replica) shard. Every partition has several shards that each host all of the data that is contained in that partition. One shard is the primary, and the others are replicas, which are redundant copies of the data in the primary shard. A primary shard is the only partition instance that allows transactions to write to the cache. A replica shard is a “mirrored” instance of the partition. It receives updates synchronously or asynchronously from the primary shard. The replica shard allows transactions to only read from the cache. Replicas are never hosted in the same container as the primary and are not normally hosted on the same server as the primary.

2.2 Caching architecture

The architecture of IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 Appliance is based on two types of servers:

- Container servers
- Catalog servers

The container servers hold the actual application data for the cache and each runs in its own JVM. The data is partitioned in such a way that each partition spans multiple containers, as shown in Figure 2-1. Each partition has one primary shard and one or more replica shards, where a shard is the smallest unit of data storage. A replica shard holds the replicas of all objects that are stored in the corresponding primary shard. Any change to an object in the primary shard is replicated to the corresponding replica object in the replica shard.

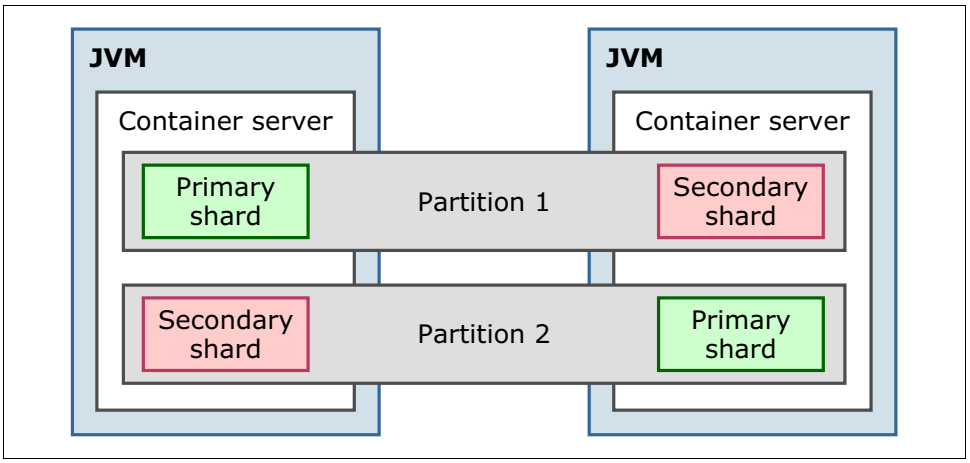

Figure 2-1 Server containers, partitions, and shards

The example in Figure 2-1 on page 17 can represent a single grid that contains a single map sets. That map set contains a single map, which is partitioned in two partitions across the two container servers. Each partition has a primary and a secondary (or replica) shard.

2.3 Configuration considerations

The critical aspects of cache infrastructure, which can help make cache design decisions easier, are now described.

2.3.1 Grids versus maps

Maps are grouped in grids. Each cache deployment can have multiple maps in a single grid or multiple grids with single or multiple maps in them. Any cache planning activity eventually touches on the question of how many grids and how many maps the cache should have. This section provides tips that can help you decide.

In the simplest case, a cache can be instantiated as one object grid and all the maps can be contained within this grid. However, in many situations there might be different requirements for different subsets of data in the grid. These different requirements might preclude those data subsets, and the maps that contain them, to be defined in separate grids. This process is because of configuration options that are defined at the level of the grid instance. You might distribute maps across different grids for the following reasons:

- Different requirements for security access control
- Different transaction timeouts that are needed for various maps

Data that is stored in the cache might require different access security levels, which can happen if one application uses heterogeneous security levels for its data. However, this case can also happen if the cache is an integration point between two or more client applications or application modules. In the multiple application environment, external applications or modules might require data access security that is enforced. And, internal applications or modules might be located close to the cache where access security control is not needed and is not wanted for performance reasons.

A grid enables the definition of the transaction timeout on all transactions that are done on the maps that are grouped in the grid. This configuration provides an option to differentiate between maps that are accessed remotely and that might need a longer time window for the transactions to complete versus maps that are accessed locally. These local transactions do not require extra time to complete.

The IBM DataPower XC10 Caching Appliance provides an extra maximum size setting for grids that can affect the number of grids that are defined on an appliance. With this setting, you can limit each grid that is defined on an appliance or collective of appliances. This feature provides more control over how the memory of an appliance or collective is used and also can imply the number of grids. For better control and usage of the memory of the appliance, you can create multiple grids, each with the capacity limited to hold different types of data or to be used by different consumers. You can also have different tenants that use an appliance or collective for which the memory usage limits must be enforced.

When you decide how many grids to declare for a specific WebSphere eXtreme Scale cache, consider the following questions:

- Does the data access require authorization?

If yes, multiple grids might be needed if access control varies between subsets of data in the grid.

If no, one grid is sufficient.

- Can all data be accessed with the same transaction timeout?

  If yes, one grid is sufficient.

  If no, multiple grids might be needed to differentiate transaction timeout for different maps.

2.3.2 Dynamic versus static maps

Backing maps that represent the server side implementation of object maps are defined in cache configuration files and are created at grid startup.

In some cases, the exact number of maps that are required later by a client application cannot be established during configuration. An example of such a situation is when the cache is used to retain monitoring data because the data might not be uniform. In addition, the data sets might be different for various time periods, and the sampling frequency would require distinct maps to retain those data sets.

Another example might be of a client application using the historical stock price information. If distinct maps are used to hold the stock price history data for each time period, then the maps would be instantiated on demand and populated based on the specific queries from users.

In general, data usage patterns might also require a different numbers of maps to hold the data. Dynamic map templates can be defined with names in the form of regular expressions. Dynamic maps are not created at the cache start time. A map instance is created only after a call to get each map is executed on a grid, and the map name matches the regular expression set as the name of the map template.

2.3.3 Number of map sets

In cases where the cache is distributed among multiple JVMs or servers, the number of map sets has to be defined. All maps grouped in a single map set are partitioned in the same way. Some applications might require that the partitioning is defined differently for various maps used by the application.

For example, the XTP application described in Chapter 5, “Enterprise caching in an extreme transaction processing environment” on page 83 uses two map sets, where one map set contains one map that is partitioned over 13 partitions, and another map set contains four maps, where each map is a container within one partition. The four maps contained within one partition are configured in this way to assure the highest speed of data access, where only one WebSphere eXtreme Scale agent queries one partition to access data from the map.

The number of synchronous and asynchronous replicas is defined for each map set. In cases where there are different failover requirements for different subsets of the data, corresponding maps might need to be grouped in different map sets. Additionally, there might be unique replication requirements for subsets of data that might be imposed by a radical difference in the rate of data change.

When evaluating how many map sets are needed for a given cache deployment, consider the following questions:

- Can all the maps in the grid be partitioned the same way?

  If yes, one map set is sufficient.

  If no, multiple map sets are required for the specific number of partitions to contain each map.
- Do all maps require the same number of synchronous and asynchronous replicas?

  If yes, one map set is sufficient.

  If no, multiple map sets are required to differentiate the number of replicas for various maps.
- Can all replicas be either read or not-read enabled?

  If yes, one map set is sufficient.

  If no, multiple map sets are required to differentiate read enablement for replicas.
- Can the computation resources be saved for all maps by setting the minimal number of active containers before shards placement process starts?

  If yes, one map set is sufficient.

  If no, multiple map sets are required to differentiate when the placement process runs for different maps.
- Can the placement strategy be the same for all maps?

  If yes, one map set is sufficient.

  If no, multiple map sets are required to differentiate placement for different maps.

2.3.4 Synchronous versus asynchronous replicas

The purpose of replicas is to increase data security and to provide failover support for the data in the cache. WebSphere eXtreme Scale allows for, but does not enforce, the use of replicas. An object grid can be created where there are only primary shards and no replicas. This configuration can be beneficiary for some development scenarios or where the data is short lived and dynamically changes. If the cache runs locally in an embedded fashion, for example in the same JVM as the application, then defining replicas for this environment does not make sense.

In most cases, the cache infrastructure is distributed and replica usage is wanted. There are two types of replicas:

- Synchronous
- Asynchronous

Even though both replica types serve the same purpose, which is to hold duplicates of data remotely for safety reasons, they each work differently in the transaction processing. They also have different impacts on the data access performance and ultimately on the application that uses the cache.

A *synchronous replica* is updated before a transaction on a primary replica is committed. This type of replica assures the highest data security at the expense of performance and is wanted where data security is more important than performance. This kind of replica is often used when the cache is the primary repository of data and where data does not change often.

An *asynchronous replica* is updated after each transaction that updates a primary shard is committed. For these replicas, the high performance is achieved at the cost of data security. If

the primary shard disappears before the replica gets updated because of a container server failure, the data is lost. This type of replica is used by applications that require extremely fast data access and where the data can be easily accessed from a traditional repository, such as a database.

2.3.5 Multi-Master Replication

Multi-Master Replication (MMR) provides replication infrastructure, which enables asynchronous data replication between multiple object grids with identical definitions. The purpose of MMR support is to assure data that is contained in multiple object grids from different catalog service domains is kept in the same state. Because any catalog service domain should not span beyond the boundaries of a single data center, MMR is used for distributed data warehousing to support multiple data centers that use WebSphere eXtreme Scale cache for data repository. Data in every center can be modified on an ongoing basis and the changes are replicated to the other data centers. In cases of change collisions, collision rules are applied.

There are a number of cache deployments that might benefit from multiple data centers infrastructure and the MMR support that increases data security and availability. One of the prominent examples is the WebSphere eXtreme Scale cache that is used for HTTP session caching of a WebSphere Application Server-based application. The distributed application uses multiple data centers to store the sessions. Usually, the application uses the closest data center to store each session for use during a client session. In a failure in the local center, the application switches to a remote data center and retrieves the replicas of the current sessions. It is proven in real life implementations that switching between data centers that are in different geographical locations is transparent to users of the application.

Multi-Master Replication imposes specific requirements on the cache configuration. The main requirement is that the grids and map sets have the same names in all catalog domains, where each catalog domain spans a single data center. All map sets must have the same number of partitions that are defined, and have the FIXED_PARTITION placement strategy in use. Additionally, all the maps and map templates must be the same across all catalog domains. The write behind and entity managers are not allowed on map sets participating in MMR.

2.3.6 Number of replicas

To increase data availability and security, increase the number of replicas. The synchronized replicas provide the highest security of data and availability but have a negative impact on the performance of the cache. Every data update is replicated to all synchronous replicas as part of the transaction and before a transaction is committed.

The higher the number of synchronous replicas that are defined for a map set, the longer it takes to complete each singular transaction on any map that is contained within the map set. Additionally, the larger the network latency that exists between the synchronous replicas and the primary, the more time it takes to replicate updates and therefore, it takes longer for the transaction to complete. For situations where the data availability and security is critical, a balance of synchronous and asynchronous replicas can be used. Asynchronous replicas are updated outside of the transaction boundaries, thus their updating does not have any direct impact on transaction execution time. For these situations, the synchronous replicas can be placed relatively local to the primaries and the asynchronous replicas can be placed remotely for better failover protection.

One of the ways to implement this scenario is with the help of Multi-Master Replication. Each grid can have a number of synchronous replicas that are limited to one and the asynchronous

replication is forwarded to a remote grid. If the performance requirements are still not met, each linked grid can have only primary shards and the data security is provided by asynchronous replication between linked primary shards from remote grids.

2.3.7 Zones

To provide high data availability and failover protection, WebSphere eXtreme Scale supports a powerful placing mechanism where shard placement within one catalog domain can be organized based on a concept of *zones*. A zone can correspond to a physical location such as a machine, a floor of a building, or an entire building.

The placement of the primary shards and their replicas can be restricted by placement rules that define where shards are placed in relation to each other. For example, you can provide robust failover protection by enforcing placement of each primary shard and its replica in different zones, where each zone is powered by a different power grid. This configuration assures that a power failure for any of the zones does not cause data loss.

Shard placements that are based on zone definitions are proven to work for zones that are defined in the scope of local machines and up to geographically distributed data centers. Nevertheless, for remote data centers, use Multi-Master Replication rather than placement based on zones.

When you define zones, consider the following questions specific to the connectivity between the zones and the data dynamics:

- What is the chance that all zones fail at the same time?
- How stable is the network connection between zones?
- What is the average network latency between zones?
- Would the average network latency assure appropriate failover protection given the rate of data change?

3

Enterprise caching in a mobile environment

This chapter describes how to take advantage of enterprise caching for a mobile environment by using the integration solution of IBM Worklight and IBM WebSphere eXtreme Scale running on a Liberty profile server. It describes the business value, integration steps, and how to develop a mobile application to take advantage of WebSphere eXtreme Scale caching.

The following topics are described:

3.1 Solution overview

In the current global enterprise business environment, with the millions of applications that run across Apple iOS, Android, Windows Phone, and other operating systems, mobile marketing brings more business opportunities for added value services. However, mobile marketing brings more challenges to existing systems.

Similar to traditional applications, mobile applications must cache large amounts of data in the memory, including session data, user state, and HTTP response. The requirements for better performance, high availability, and linear scalability are critical to mobile services. Improving the user experience is the key issue that is faced by mobile businesses.

IBM Worklight is part of the IBM Mobile Foundation and provides an open, comprehensive, and advanced mobile application operating system. *WebSphere eXtreme Scale* is one of the caching infrastructures for the WebSphere family of products that provides distributed object caching to help manage application data and business logic for multiple servers. Without complex configuration and with only minimal code changes, mobile applications can benefit from the WebSphere eXtreme Scale cache seamlessly. By integrating these elastic cache capabilities with IBM Worklight, you can accelerate mobile businesses and provide a better experience for the user.

In this example solution, both the IBM WebSphere eXtreme Scale servers and IBM Worklight servers use WebSphere Application Server Version 8.5 Liberty profile servers. The *Liberty* profile is a composable, lightweight runtime application server. WebSphere eXtreme Scale features can be enabled quickly and easily by using the Liberty profile.

This solution uses WebSphere eXtreme Scale client to access remote WebSphere eXtreme Scale grids that are then started by the Worklight HTTP adapter. As such, this solution focuses on using the WebSphere eXtreme Scale as a side cache.

Figure 3-1 shows an overview of the solution architecture.

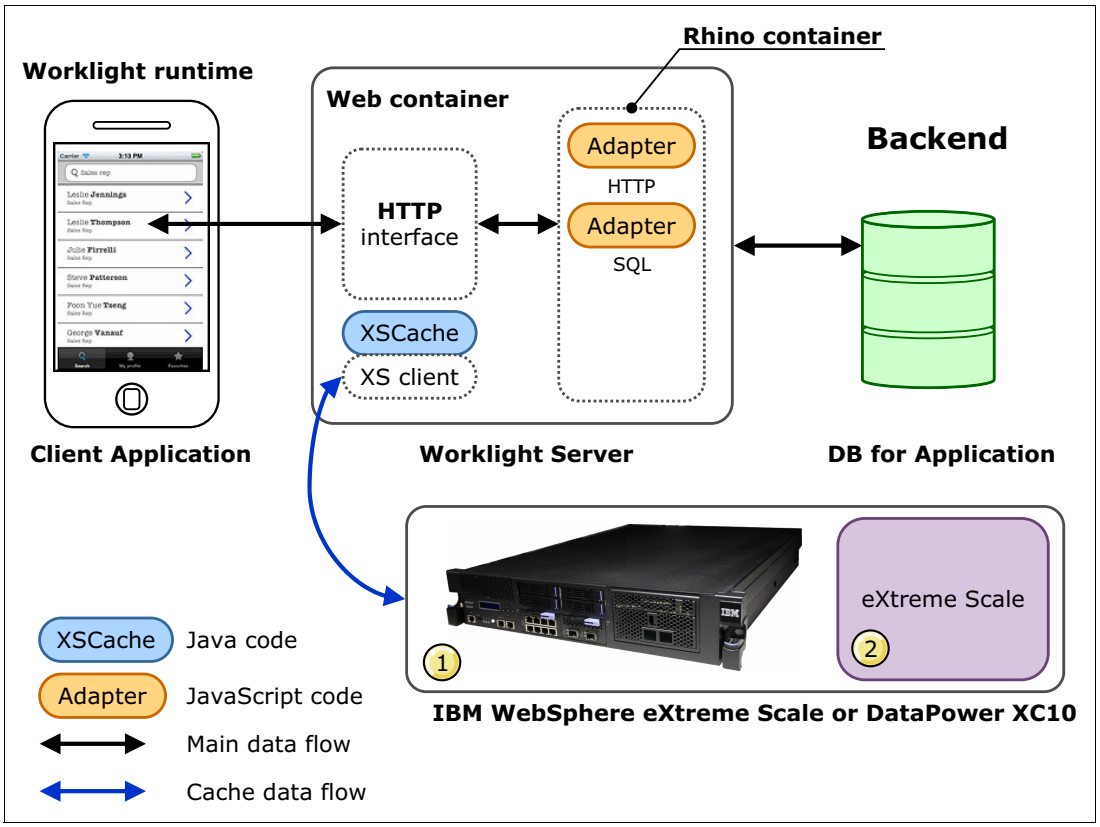

Figure 3-1 Solution architecture

3.2 Integration of IBM Worklight and WebSphere eXtreme Scale

This section describes how to run the IBM Worklight server and IBM WebSphere eXtreme Scale server by using the WebSphere Application Server V8.5 Liberty profile server. It includes the configuration of the Liberty profile server, WebSphere eXtreme Scale, and IBM Worklight.

3.2.1 Integration overview

Because IBM Worklight and WebSphere eXtreme Scale can run on different servers, you have your choice of different combinations when you set up your environment.

You can use IBM Installation Manager to install WebSphere eXtreme Scale in one of the following configurations:

- Integrated with WebSphere Application Server
- In a stand-alone environment

Currently, you can install the WebSphere Application Server V8.5 Liberty profile by using one of the following options:

- Option 1: Install the Liberty profile feature when you install the WebSphere Application Server V8.5. In this option, you must integrate WebSphere eXtreme Scale with WebSphere Application Server.

- Option 2: Install from a single JAR file. (Currently, the file name is `wlp-nd-8.5.0.0.jar` or `wlp-developers-8.5.0.0.jar`.) When you use this option, you must install the WebSphere eXtreme Scale for Developers - Liberty Profile from the `wxs-wlp_850.jar` file. This file is available through the IBM developerWorks® site:

  http://www.ibm.com/developerworks/downloads/ws/wsdg/

When you use IBM Installation Manager to install the IBM Worklight server, you have multiple options for the application server and the database management system. In the example scenario that is used in this book, IBM Worklight is installed by using the WebSphere Application Server V8.5 Liberty profile using Red Hat Enterprise Linux.

Installing the needed products

This solution includes the following products:

- WebSphere eXtreme Scale V8.5
- IBM Worklight V5.0

Ensure that these products are already installed. If not, some information about product installation is provided in Chapter 6, “Installation and configuration” on page 103. More details for installation can be found in the information centers for each of the products.

3.2.2 Running IBM Worklight server on a Liberty profile

When IBM Worklight was installed, following the instructions in 6.4, “Installing IBM Worklight 5.0” on page 124, the Liberty profile server was installed and configured to be used as the application server run time for the IBM Worklight server. This section describes extra configuration that is needed and how to run the IBM Worklight server on the Liberty profile server.

Understanding the Worklight server configuration

When the IBM Worklight Server is installed with the Liberty profile, the installation program completes the initial configuration steps that are required to run the Worklight server. In the `server.xml` file for Worklight server, as shown in Example 3-1, the installation program defines the necessary features and Worklight applications. It also defines a basicRegistry for user authorization and authentication.

Example 3-1 Example of the server.xml file for Worklight server

```
<server description="worklight server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>ssl-1.0</feature>
        <feature>servlet-3.0</feature>
        <feature>jdbc-4.0</feature>
        <feature>security-1.0</feature>
        <feature>appSecurity-1.0</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" >
      <tcpOptions soReuseAddr="true"/>
```

```
    </httpEndpoint>

    <application id="worklight" location="worklight.war" name="worklight"
      type="war">
            <classloader delegation="parentLast">
              <commonLibrary>
                <fileset dir="${shared.resource.dir}/lib" includes="*.jar"/>
              </commonLibrary>
            </classloader>
    </application>


    <application id="applicationcenter" location="ibmapplicationcenter.war"
      name="applicationcenter" type="war">
            <application-bnd>
                <security-role name="appcenteruser">
                   <group name="appcentergroup"/>
                </security-role>
            </application-bnd>
    </application>

    <!-- Declare the user registry for the IBM Application Center. -->
    <basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
        <!-- The user "appcenteradmin" has special privileges within the IBM
         Application Center. -->
        <user name="appcenteradmin" password="admin"/>
        <!-- The other users have normal privileges. -->
        <user name="demo" password="demo"/>
        <group name="appcentergroup">
            <member name="appcenteradmin"/>
            <member name="demo"/>
        </group>
    </basicRegistry>

</server>
```

Enabling WebSphere eXtreme Scale client on Worklight server

As installed, the IBM Worklight server does not use WebSphere eXtreme Scale. To enable the WebSphere eXtreme Scale client feature in the Liberty profile server, add the following line to the `featureManager` element in the `server.xml` file, which is on UNIX systems in the *<WORKLIGHT_HOME>*`/server/wlp/servers/worklightServer` directory:

```
<feature>eXtremeScale.client-1.0</feature>
```

The location of the `server.xml` file and logs for the Liberty profile are in a different location on Windows operating systems. If you are running on Windows, check the product documentation for the location of logs and configuration data for your Windows edition.

Starting the Worklight server

Run the following command from the *<WORKLIGHT_HOME>*`/server/wlp/bin`:

```
server start worklightServer
```

where *<WORKLIGHT_HOME>* is the IBM Worklight installation directory.

Figure 3-2 is an example of the worklightServer started confirmation to verify that the installation succeeded.

```
[root@localhost bin]# ./server start worklightServer
Server worklightServer started with process ID 19170.
[root@localhost bin]#
```

Figure 3-2 Verify IBM Worklight Server started

You can also verify that the Worklight Server is started by accessing the IBM Worklight Console, which is shown in Figure 3-3, from a web browser by accessing the following location (where *<server_host_name>* is the host name where IBM Worklight Server is installed and started):

`http://<server_host_name>:9080/console`

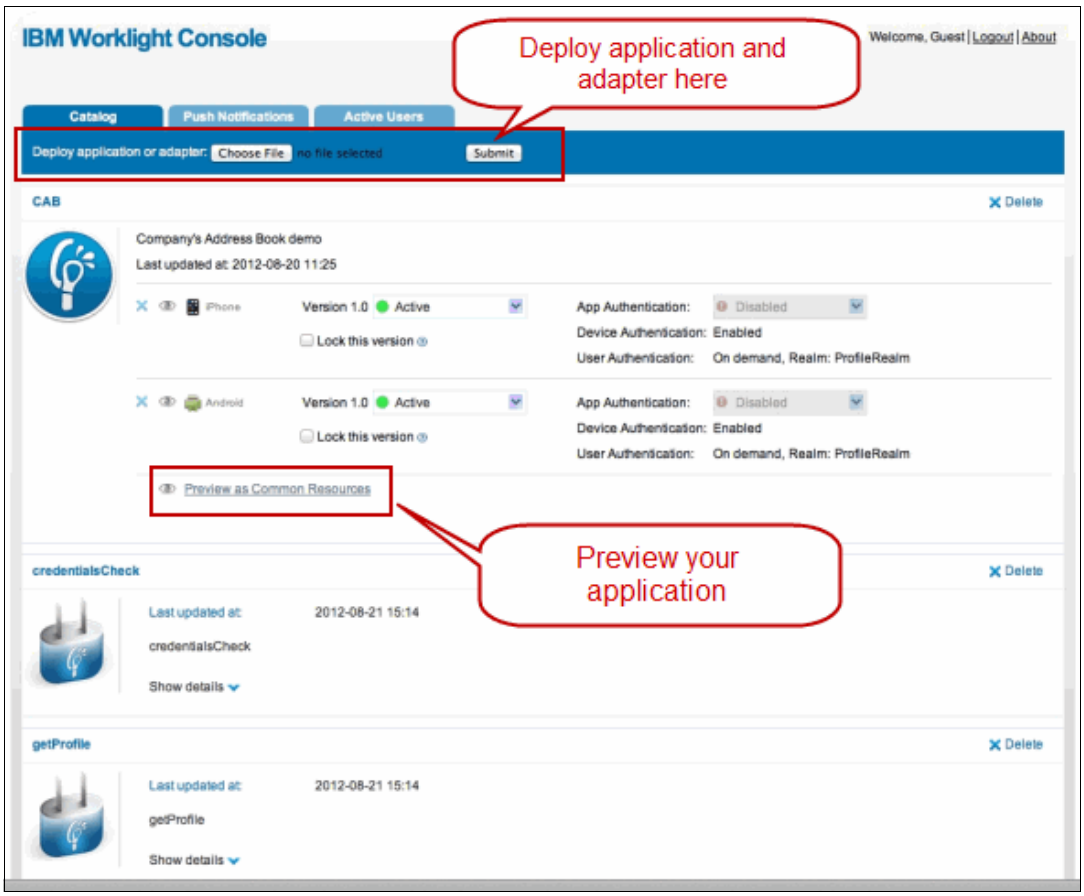

Figure 3-3 The IBM Worklight console

With the Worklight console, you can do the following tasks:

- Deploy a Worklight application
- Deploy a Worklight adapter
- Preview and test your Worklight server-side application with a browser

3.3 Integrating IBM Worklight and the XC10 appliance

You can also use the DataPower XC10 Caching Appliance as the caching technology in a mobile solution. To use a DataPower XC10 as your cache store, you must only change the catalog service endpoint and switch to the grids and maps that are created from the XC10 console.

The XC10 Caching Appliance is typically used as one of the following types of cache:

- Simple Data Grid for ObjectMap application programming interface (API) usages
- Dynamic cache mainly for web content cache
- Session cache for HTTP session data cache

3.4 Using caching in mobile applications

How to develop a mobile application that uses a cache is now introduced.

3.4.1 Worklight application overview

When you consider mobile solutions or mobile applications for enterprise, keep in mind the following implementation considerations:

- What kind of mobile application will you provide (for example, a game or an online shopping store)?
- Should you choose basic application, web application, or both?
- Will you support online mode, offline mode, or both?
- What mobile devices will you support?
- How will you develop and deploy your application?

IBM Worklight is the solution for enterprise mobile applications. It provides an Eclipse-based integrated development environment (IDE) that can run on multiple operating systems. It supports the development of basic, web, and hybrid applications for iOS, Android, Windows Phone, and BlackBerry. You can deploy, manage, and run mobile applications on the Worklight server. It supports the following web technologies:

- JavaScript
- Ajax
- HTML5
- CSS

A typical Worklight application topology looks similar to what is shown in Figure 3-4 on page 30 and includes a three-tier architecture with a client, a Worklight Server, and a back-end system, which might be a database, a service bus, or a back-end service.

Figure 3-4 shows a Worklight application overview.

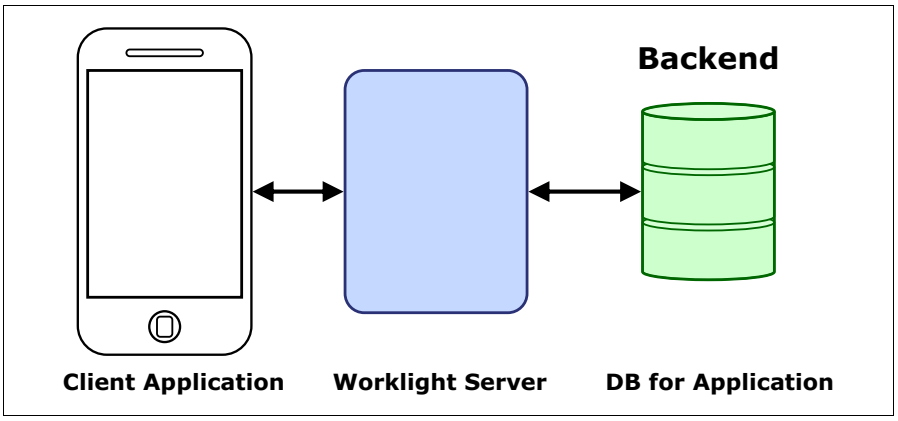

Figure 3-4 Worklight application overview

In this solution, the client communicates with the Worklight server through the HTTP protocol and then the Worklight server uses Java Database Connectivity (JDBC) to communicate with the back-end system.

For the client side, web technologies are used to develop the Worklight client application. It is deployed on the consumer device (such as a smartphone or tablet). Worklight provides a JavaScript framework to facilitate the development. The main API is `WL.client`. Most client logic is implemented in JavaScript, which also allows the generation of unique basic assets for various mobile operating systems, such as iOS, BlackBerry, Windows Phone, and Android. Developers can also take advantage of basic device capabilities, such as the camera, GPS, and accelerometer.

On the server side, Worklight provides an HTTP interface to handle the HTTP requests. If the application must communicate with a back-end system, a Worklight adapter is used. Adapters are typically written in JavaScript and run in the Rhino container. Along with adapters, you can also run other server-side code that is called *customizations*, which are written in Java. The Rhino container lets you start Java code from the JavaScript, meaning your adapter can call your customization. This solution uses this capability.

Figure 3-5 provides an overview of these components.

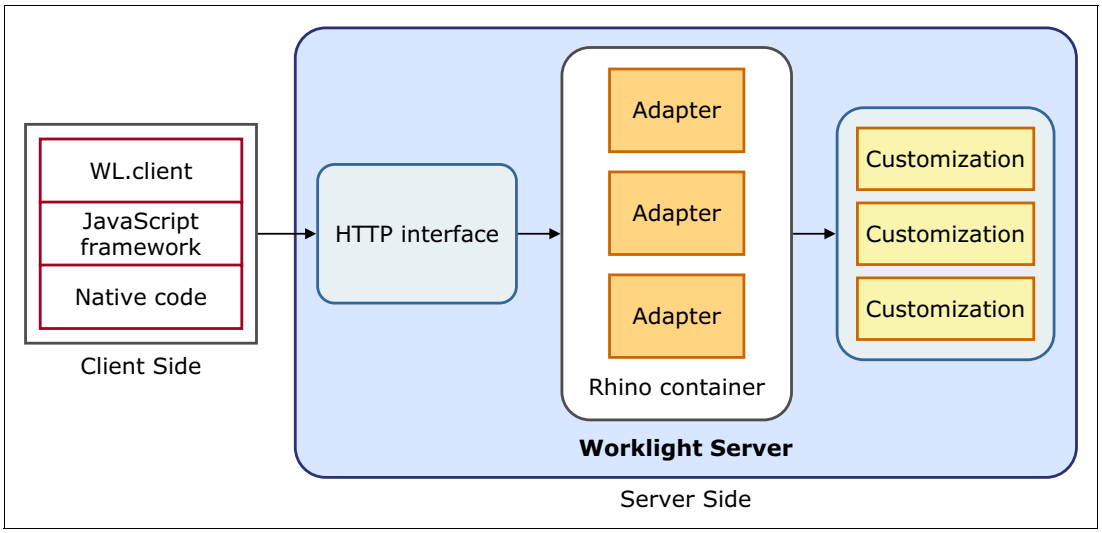

Figure 3-5 Components for the Worklight application

3.4.2 Developing your mobile application

Instructions for using Worklight to develop your mobile application are now provided.

To develop the mobile application, a mobile developer uses IBM Worklight Studio as the IDE. IBM Worklight Studio can be installed on various operating systems including Windows or Mac OS X, following the instructions that are provided in 6.4.2, “Installing IBM Worklight Studio” on page 129. For this scenario, IBM Worklight Studio on Mac OS X is used.

> **IBM Worklight Studio on a Mac:** If you are developing basic mobile applications for iOS, it is suggested that you use IBM Worklight Studio on a Mac, because you must use a Mac to build iOS assets.

Working with a Worklight project

To develop a mobile application with Worklight Studio, take the following steps:

1. Create a Worklight project. Figure 3-6 shows an example of the Company Address Book (CAB) mobile application project.

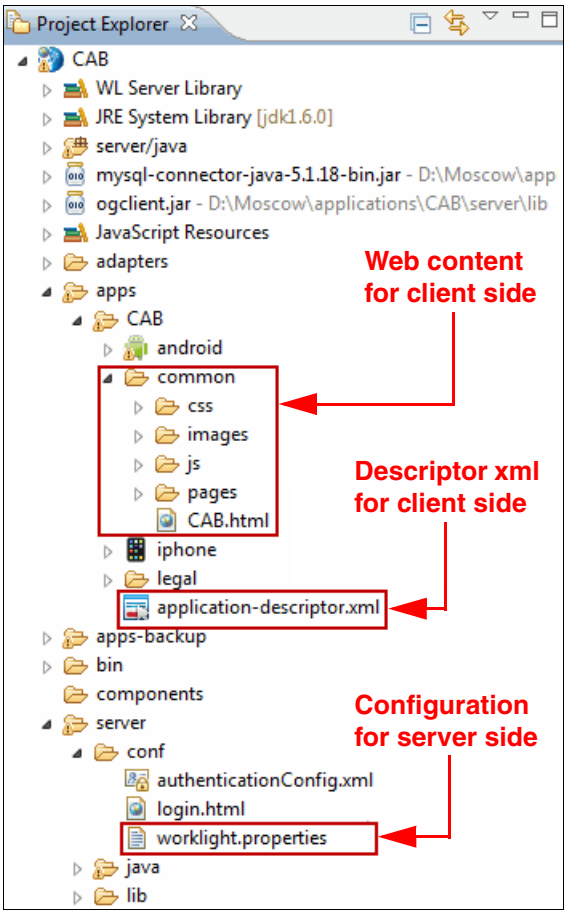

Figure 3-6 A typical Worklight project

2. Next, you develop your mobile application. The web content for the client side is under the `apps/<your_app_name>/common` directory and the logic is implemented by using JavaScript code. As you can see, the application for this scenario is already developed.

3. To generate the artifacts for a specific device, click the **Worklight** icon on the toolbar to display the menu and then select **Worklight Environment**, as shown in Figure 3-7.

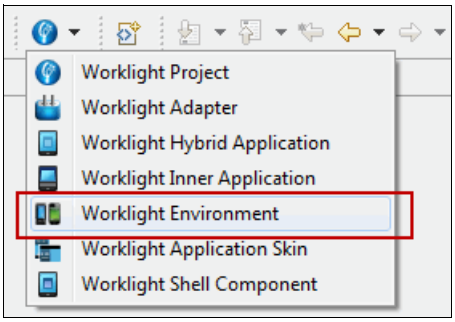

Figure 3-7 Using the Worklight Environment functionality

4. Choose the platforms that your application supports, as shown in Figure 3-8. The appropriate artifacts for your selected platforms are generated and stored in your project.

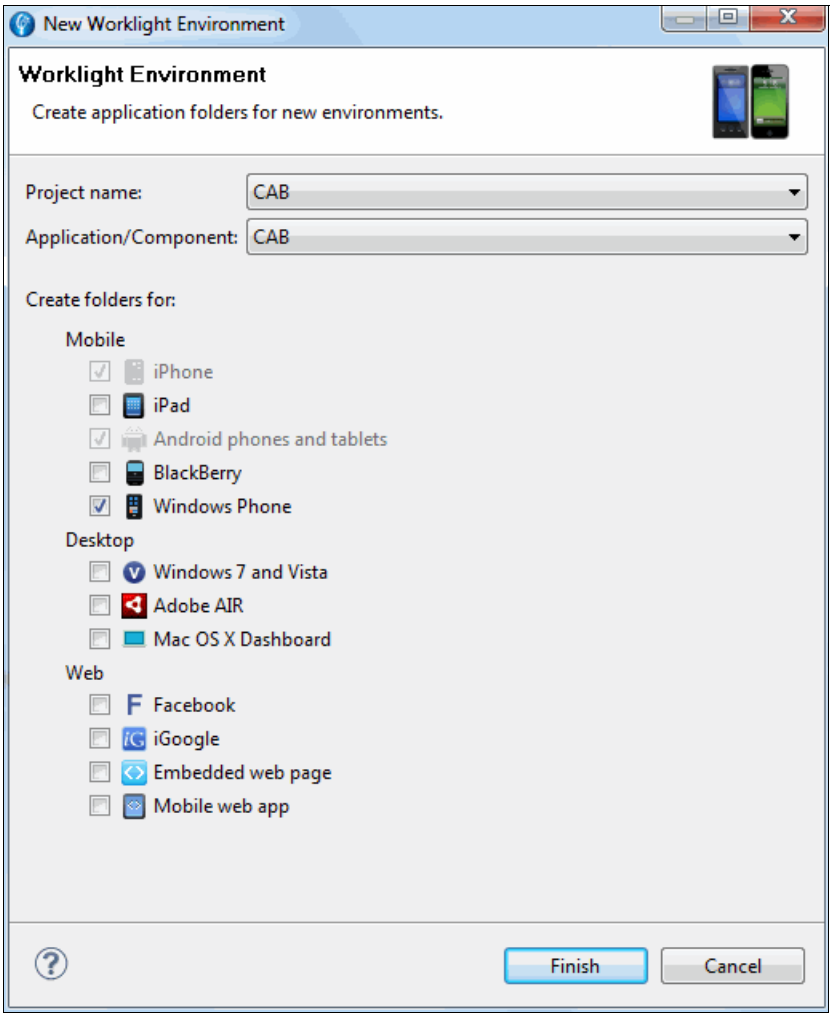

Figure 3-8 Selecting the target mobile device for your application

5. Next, build and deploy your mobile application for testing by using a device simulator. Select a target device (in this example, an iPhone), right-click the device, and select **Run As** → **Build Environment and Deploy**, as shown in Figure 3-9.

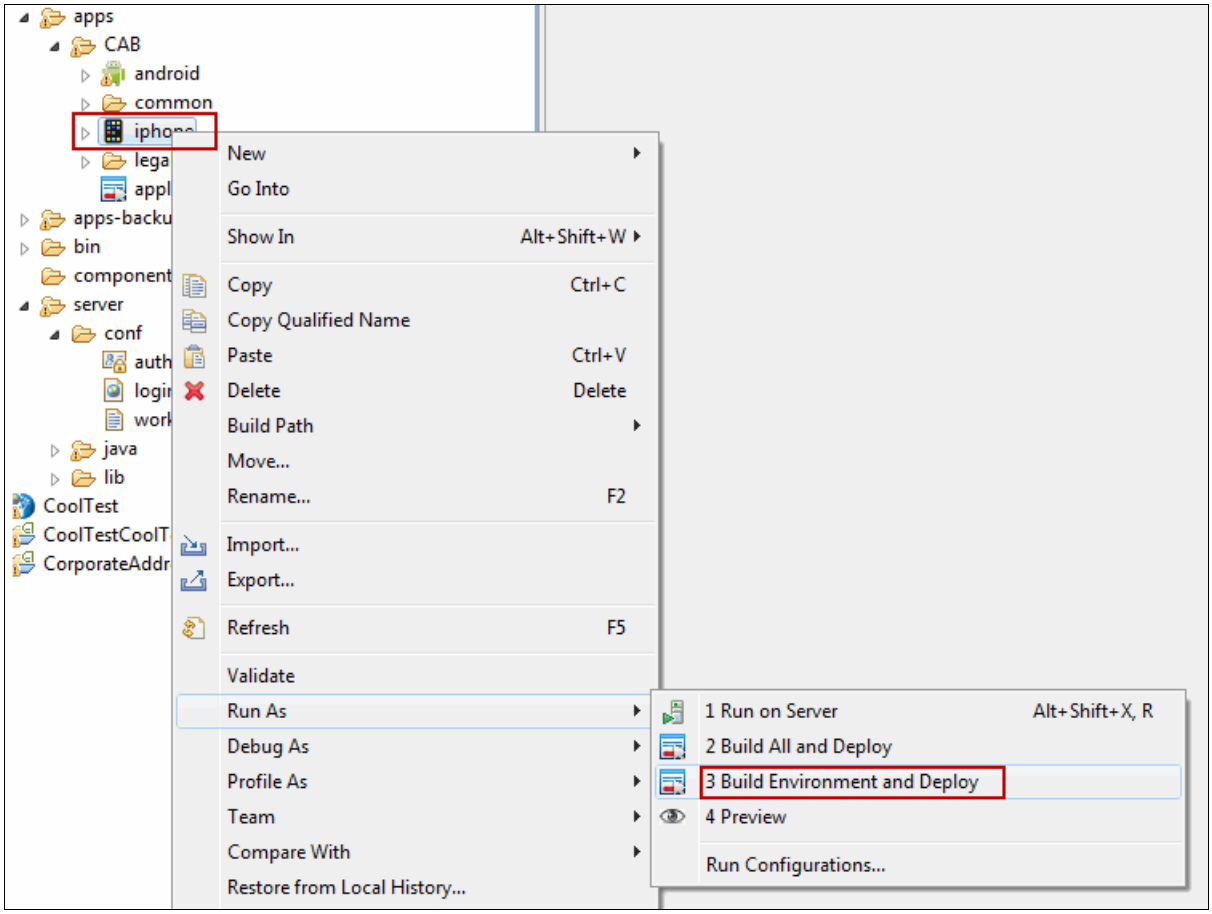

Figure 3-9 Building a specific project for a target device

This generates a new platform-specific project in the workspace.

6. To test your mobile application with a device simulator, select the generated project and click **Run As** → <***your device specific project***>. It starts the device simulator with your mobile application so that you can test the functionality.

3.4.3 Reasons to use caching with a mobile application

Caching requirements for mobile applications are similar to traditional applications, but there are some requirements that are mobile-specific:

- Mobile applications use more volatile data in memory.
- The back-end system can become a bottleneck when the workload is heavy and impact response times, which tend to be more critical in mobile applications. Caching can relieve the pressure to the back-end system.

Consider the following specifics carefully for mobile applications:

- Extremely high and unpredictable workloads because the applications can be accessed at any time and from anywhere
- Frequent peak times

- Interaction data size is smaller, but more frequently requested
- The need to efficiently generate and parse JavaScript Object Notation (JSON) data

For example, suppose there is a mobile application that follows sporting events. You can predict heavy loads during popular sporting events. The results of a game and specific information about the scoring players, for example, are in high-demand and the same for all users at a specific point-in-time during the game. The final results and information of a game do not change after the game finishes. This type of data is an excellent candidate for caching. In the case of final results, after the final results are in the cache, all requests pull the data directly from the cache and never have to do a request to the back-end data store. For data that is updated, like scores during a game, having the cache loaded by a service as the data changes allows the cache to have the most up-to-date information and handle all of the incoming requests. In this way, if you have a million requests for the score of a game, the database does not have to handle a million requests, which would cause performance issues with the database. Both IBM WebSphere eXtreme Scale and the DataPower XC10 Caching Appliance are effective *middle men* and can improve response times, leading to overall client satisfaction and improved business results.

With WebSphere eXtreme Scale, the access path to the back-end system can be reduced. Caching the JSON data in the grid eliminates the expense of JSON generation, which can also help to improve performance.

3.4.4 Adding caching to the Company Address Book scenario

This scenario uses a simple mobile application called *Company Address Book* (CAB), shown in Figure 3-10. From the main window, the user can search the address book by attributes such as given name, surname, title, and phone number, and is able to search for any information that is contained in the address book back-end application. The result of each search is displayed as a list of records that satisfies the search criteria. The user can drill down to view a person's data, including a photo of the person. The application reads all of this data from the company's back-end database.

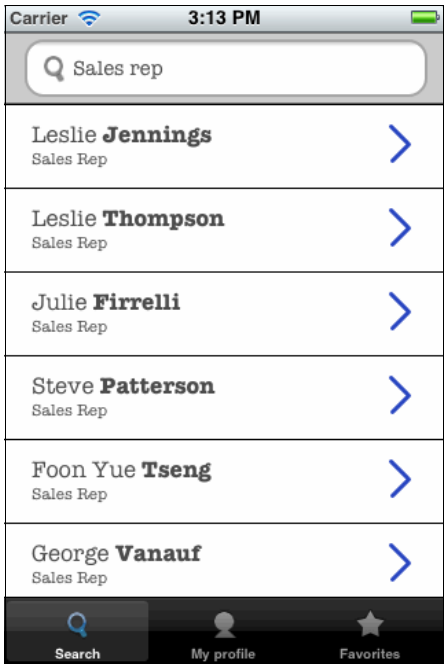

Figure 3-10 CAB mobile application

After appropriate authentication and authorization checking, the address book application also allows a user to edit profile information, including adding a picture from the local device camera. The application also supports storing encrypted contact data locally on a device.

The integration of the caching solution with this mobile application is in support of caching the search results. The same search might be done multiple times by different people, such as searching for certain sales representatives in the organization or for the CEO. By caching the results of searches, the database does not have to be queried again for the same search request. This process not only reduces the response time of the search itself, but also reduces the load on the back-end database.

The CAB application includes the following components:

- Generic code that is common to all operating systems
- Basic code for iOS and Android. This code is used to implement Human Interface Guidance (HIG) related functions for Apple and Google.
- The following adapters:
  - Server-side code for connectivity
  - The main search adapter, with search, specific search, and authentication functions
  - Check for My profile functionality
  - getProfile adapter for My profile functionality

Figure 3-11 shows the Worklight project on Mac OS X within IBM Worklight Studio. It contains an embedded Worklight server with external libraries to connect to IBM WebSphere eXtreme Scale and the back-end database.

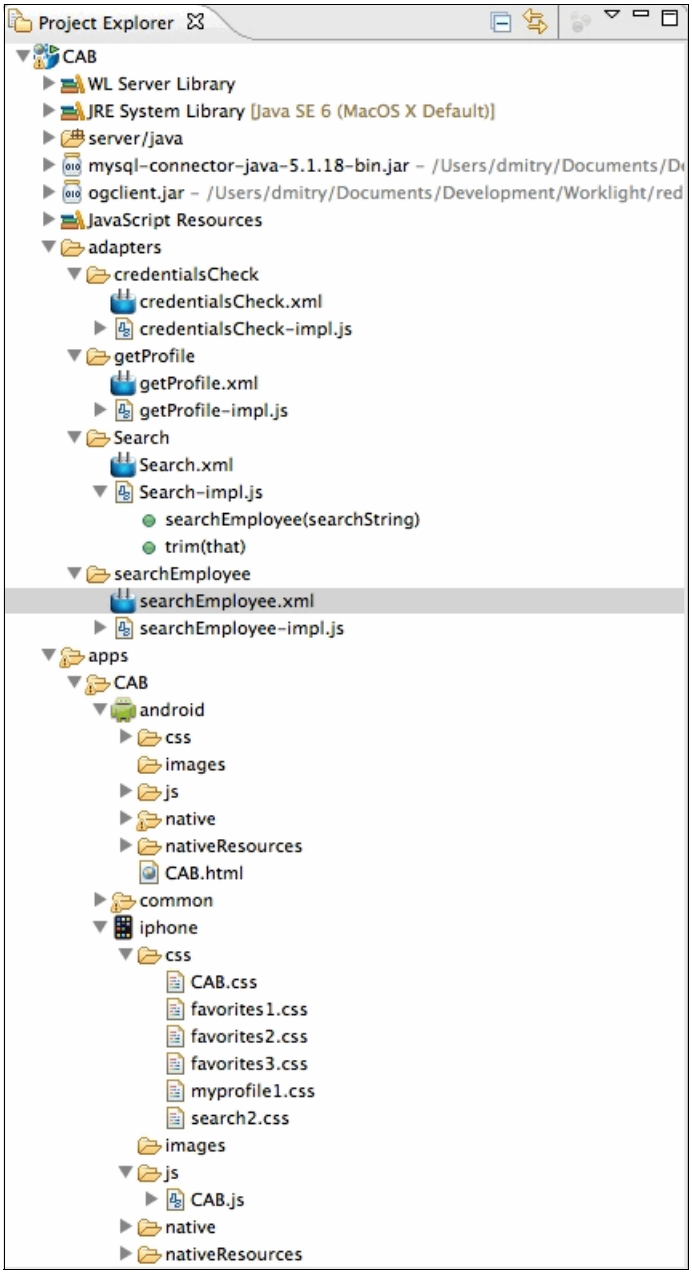

Figure 3-11 Project structure on Mac OS X

All requests from a mobile device go to the main search adapter. It parses the request and is started by other adapters on behalf of the user. The main data flow is shown in Figure 3-12.

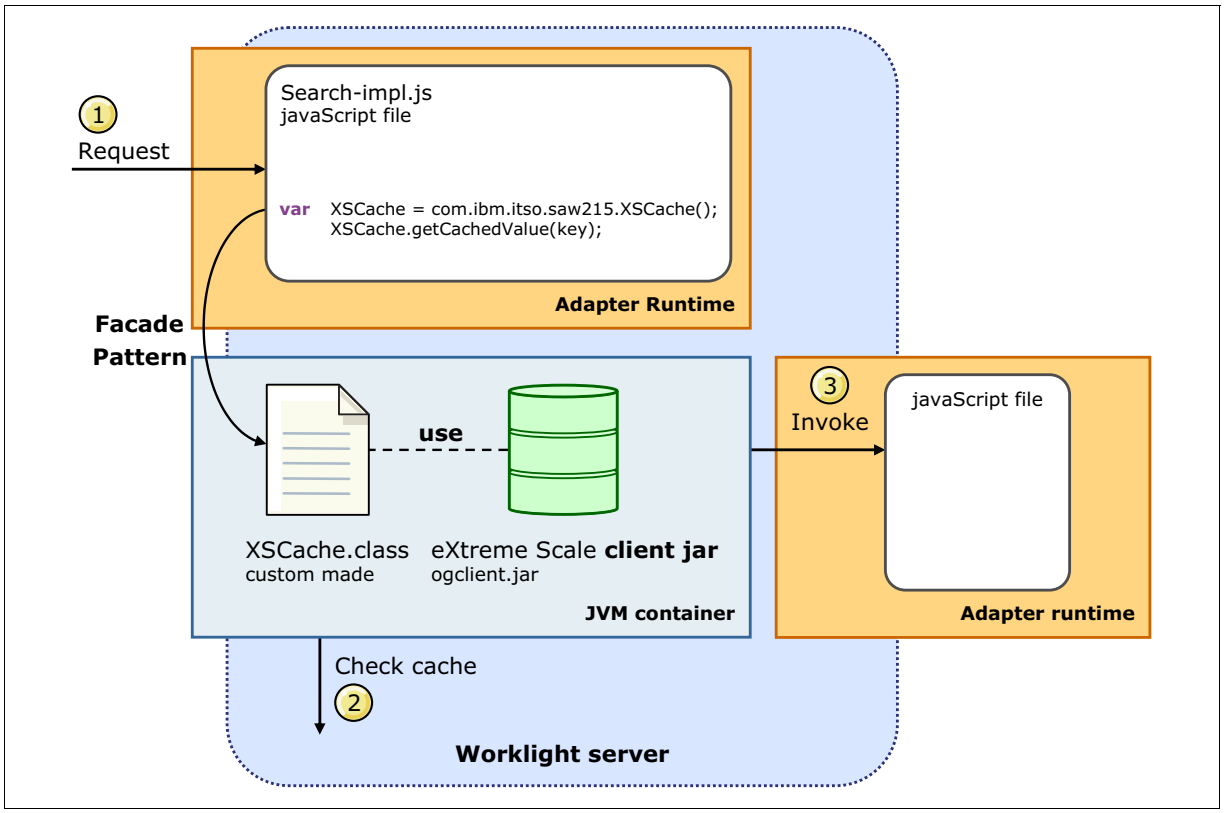

Figure 3-12 Application workflow using eXtreme Scale

To use caching, a simple extension class is written in Java. The class acts as a proxy and every request for data passes through the class. The cache is checked for data, and, if found, the data is returned from the cache. Otherwise, the class starts an adapter to retrieve the data from the database, stores this data in the cache, and returns it to the main adapter. The data is then sent back to mobile application for displaying it to the user.

The customization class works with both WebSphere eXtreme Scale and DataPower XC10. All configuration data is stored in the Worklight configuration files and the cache server addresses can be changed dynamically without recompiling the code.

3.4.5 Using caching in the address book application

How caching is implemented for the CAB application is now described.

Preparing to use caching

In this scenario, IBM WebSphere eXtreme Scale is installed on a server and the WebSphere eXtreme Scale client runtime JAR files must be copied from the eXtreme Scale server and then imported into the IBM Worklight Studio mobile application project. This process allows the mobile application to connect to WebSphere eXtreme Scale.

The WebSphere eXtreme Scale client run time, `ogclient.jar`, is in the *<XS_HOME>*/lib directory on the IBM WebSphere eXtreme Scale server. To import it into your IBM Worklight project, follow these steps:

1. Expand your project in the Project Explorer so that you can see the lib folder under the server folder.
2. Right-click the server/lib folder in your project, and select **File** → **Import**.
3. In the Import dialog, expand **General** and select **File System**. Then, click **Next**.
4. Click **Browse** and go to the location of your `ogclient.jar`. Select the file and click **Finish**.
5. Verify that the `ogclient.jar` file is now shown under the lib folder.

Implementing the customization with Java code

A custom class is used to minimize changes in the existing Worklight project to implement the caching solution. In this way, you can create an application without using the cache and then, with only a minor change, can have it working with the cache. This process is a scenario where using the Facade Design Pattern can help you.

The *Facade Design Pattern* is a way to *wrap* a complicated subsystem with a simpler interface. For the Company Address Book mobile application, the XSCache class provides this facade. All of the API interactions are done within this class and display a single method, getCachedValue(), to the Worklight Server. This method is shown in Example 3-2 and the complete class is shown in “Additional files for Chapter 3” on page 135.

Example 3-2 getCachedValue() method in XSCache.java

```
public Object getCachedValue(String key) {
   Scriptable result = null;
   //key can't be null
   if (key == null) {
      return null;
   }

   ObjectMap map ;
   Object obj;
   try {
         connect();
         map = session.getMap(this.mapName);
         obj = map.get(key);
         if (obj != null) {
             //return a cached data
            logger.info("Found " + key + " from Grid");
            String data = (String)obj ;
            JSONObject jo = JSONObject.parse(data);
            result = JSObjectConverter.jsonToScriptable(jo);
            logger.info("Get " + key + " from Grid");
            disconnect ();
            return result;
         else {
            //invoke a WL procedure to get data
            DataAccessService das =
                  WorklightBundles.getInstance().getDataAccessService();
            ProcedureQName pq = new ProcedureQName(this.adapterName,
                  this.procedureName);
            InvocationResult invResult = das.invokeProcedure(pq,
```

```
                    "['" + key + "']");
            JSONObject jsonObject = invResult.toJSON();
            result = JSObjectConverter.jsonToScriptable(jsonObject);
            //cache result in XS
            if (jsonObject != null &&
               jsonObject.get("isSuccessful").toString() == "true")){
               map.insert(key,jsonObject.serialize());
               logger.info("Succesfully put " + key + " in Grid");
            }
      } catch (ObjectGridException e) {
            logger.info("ObjectGridException: Error while putting " + key +" in
                        Grid");
            e.printStackTrace();
      } catch (IOException e) {
            logger.info("Can't serialize object. Error while putting value in
Grid");
      } catch (TransactionCallbackException e) {
            return invokeWLAdapter(key);
      } finally {
            disconnect();
      }
      return result;
   }
```

The main design point is what to use as a key in your cache. For this implementation, the key is what the mobile application user provides in the search field of the application, which is a parameter in Worklight adapter invocation. For example, for a search request for “Sales rep”, the key is “Sales rep”.

The logic in this public method first checks the key to verify that it is not null because the cache does not support a null key. After it confirms a non-null key, the value for the key is retrieved from the cache by using the get() method of the ObjectMap. If the value returned is not null, this means you have a *cache hit* and the method returns the value from the cache after converting it to the correct JSON representation for JavaScript. If a null is returned from the get() method of the ObjectMap, it is a *cache miss,* which means that the requested data does not currently exist in the cache. In this case, the method must retrieve the data from the back-end database by starting the Worklight Adapter by using the Worklight Java invocation API. If this process is successful, the method adds the information into the cache by using the insert() method and returns the data back to the main adapter that requested it. It caches only the result if the request from the back-end database was successful, so the class does not cache network connection errors or timeouts.

The mobile application solution should always process errors gracefully. If there is a network error while connecting to WebSphere eXtreme Scale or the cache is unavailable, the application continues to work without error.

The class also builds in flexibility within the constructor method. The constructor is designed so that every adapter that uses the cache can access a separate map.

To integrate this class, add the Java file for the class to the `server/java` folder in your Worklight Studio project.

Starting cache in Worklight HTTP adapter

The main adapter that receives all requests from the mobile application is named *Search*. There are two files in the Worklight project, a descriptor file named `Search.xml` and an implementation file named `Search-impl.js`.

The `Search-impl.js` file is the only place in the project that is modified to integrate the caching into the mobile application. The full file contents is shown in "Additional files for Chapter 3" on page 135. The searchEmployee() function, which is shown in Example 3-3, is where the cache wrapper class, XSCache, is called to get the cached value.

Example 3-3 searchEmployee() function in Search-impl.js

```
function searchEmployee(searchString) {
      var splitArray,resultArray = new Array();
      var searchResult,success = false;
      var serverName = WL.Server.configuration["cache.server.name"];
      var serverPort = WL.Server.configuration["cache.server.port"];
      var gridName = WL.Server.configuration["cache.grid.name"];
      var mapName = WL.Server.configuration["cache.map.name"];
         var splitArray = searchString.split('OR');
   //insert this for using eXtreamScale
      var XSCache = com.ibm.itso.saw215.XSCache(serverName, serverPort,
            gridName, mapName, "searchEmployee", "search");
   var i;
   for (i = 0; i < splitArray.length; i++) {
      searchResult = XSCache.getCachedValue(trim(splitArray[i]));
      //original code without eXtreamScale
      /*
       searchResult = WL.Server.invokeProcedure({
         adapter:'searchEmployee',
         procedure:'search',
         parameters:[trim(splitArray[i])]
      });
      */
      if (searchResult !== null) {
         success = searchResult.isSuccessful;
         resultArray = resultArray.concat(searchResult.resultSet);
      }
   }
   //return searchResult;
   return {"isSuccessful":success, "resultSet": resultArray};
}
```

An interesting design point of this code is that the configuration of the connection to the cache server is stored in the `worklight.properties` file and it instantiates the XSCache class that is based on these parameters:

```
serverName = WL.Server.configuration["cache.server.name"],
serverPort = WL.Server.configuration["cache.server.port"],
gridName = WL.Server.configuration["cache.grid.name"],
mapName = WL.Server.configuration["cache.map.name"],
```

By writing this wrapper class generically, and taking all of the specific details, such as the server name, port, grid, and map, the same class can be used for other adapters as well without any code modifications.

The original code (without WebSphere eXtreme Scale) is shown in Example 3-4.

Example 3-4 Original code to start search from back end

```
searchResult = WL.Server.invokeProcedure({
   adapter:'searchEmployee',
   procedure:'search',
   parameters:[trim(splitArray[i])]
});
```

The `searchEmployee` adapter is called with the procedure `search` along with the search parameters.

To use the cache, you instantiate the XSCache class from JavaScript and pass in the adapter name, `searchEmployee`, and the procedure, `search` as parameters.

```
var XSCache = com.ibm.itso.saw215.XSCache(serverName, serverPort, gridName,
        mapName, "searchEmployee", "search");
```

The search parameters are passed in to the getCachedValue() method:

```
searchResult = XSCache.getCachedValue(trim(splitArray[i]));
```

The XSCache class handles all of the work to determine if the value is in the cache and if not, to retrieve the data from the back end and add it to the cache.

The information for the cache itself (server, port, grid, and map) is read from the `worklight.properties` file. Add the following configuration parameters for connecting to the cache server:

```
cache.server.name=<your XS cache server ip address, example '180.180.120.10'>
cache.server.port=<your XS cache server ip address, example '2809'>
cache.grid.name=<your XS cache grid name, example 'Grid'>
cache.map.name=<your XS cache map name, example 'Map1'>
```

This configuration can be easily modified to use an XC10 appliance as the cache instead of IBM WebSphere eXtreme Scale. Change the properties file to have the same grid name and map name on the XC10 appliance. For the server name and port, use the `catalogServiceEndpoint` from the XC10 console, splitting the `catalogServiceEndpoint` into the server name/IP address and port number with the colon (:) delimiter.

3.4.6 Deployment

Figure 3-13 shows the topology that is used in this solution, which has three catalog servers and four container servers. Use one Worklight server as shown, and you can scale out to use a Worklight cluster for your product system.

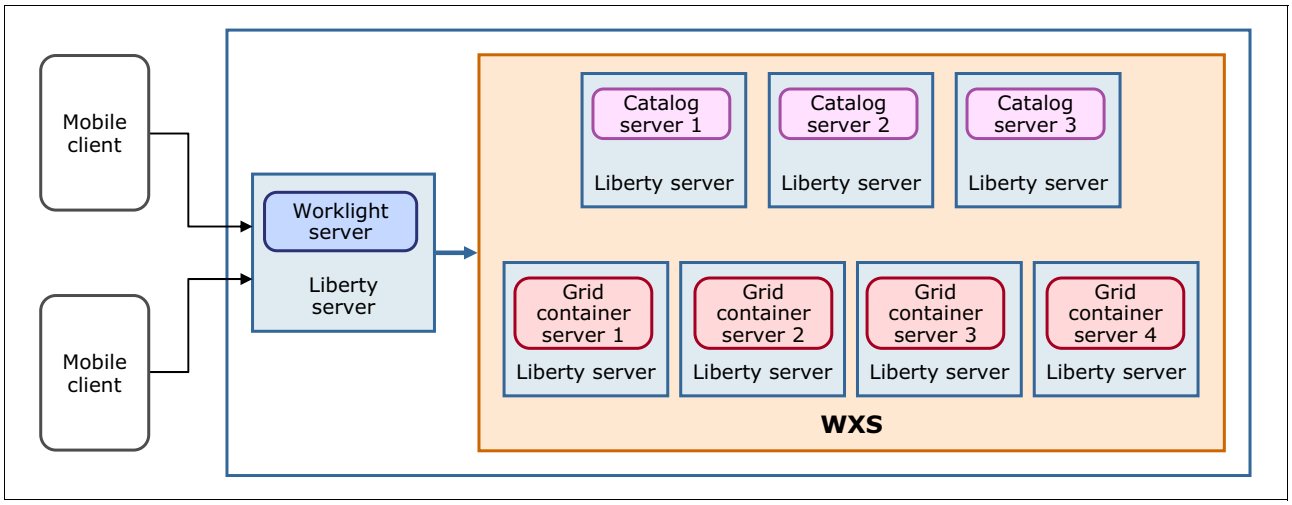

Figure 3-13 The deployment topology

Typically, a Worklight application includes the following types of deployable artifacts:

- Worklight application: File extension ends with `.wlapp`
- Worklight adapter: File extension ends with `.adapter`
- Worklight customization: File extension ends with `.jar` or `.war`. This solution uses the `.jar` extension.

Mobile applications typically run in the following environments:

- Development
- Test
- Production

In the development environment, Worklight Studio can help to build all of the deployable artifacts automatically and deploy the application to the embedded Worklight server.

In the test and production environments, in addition to using the Worklight console for deployment, you can use the Worklight Ant tasks for the build and deployment of the mobile application. “Sample Worklight ant build script” on page 138 shows an example Ant script for build and deployment:

3.4.7 Debugging and testing the cache

After the implementation to use the cache with the mobile application is completed, it is important to test the mobile application and, if problems are found, be able to debug the problems. Methods to test the cache in a mobile application are now described.

Logging

You can use logging as a method to debug and test the mobile application. For example, you can use logging to test the connection with eXtreme Scale grid containers. Set `showLogger` to `true` in `WL.client.init()` and then use the `WL.Logger.debug()` and `WL.Logger.error()` methods for the development and test environment.

For the iPhone and iPad, the message is sent to the debugger console of *Xcode*, which is the development environment for iPhone and iPad. For Android, the message is printed in the Android *LogCat*, which is accessible in the Android development environment. For BlackBerry, the message is printed to the BlackBerry *event log*.

Preview as Common Resources window

From the IBM Worklight Console, as shown in Figure 3-3 on page 28, click **Preview as Common Resources** to open a new window. From this window, you can test your application without starting the simulator. You can verify the HTTP communication in a browser by using a debugging tool such as Firebug. This method is an easy way to debug your JavaScript and verify whether the client gets correct JSON data from the cache.

The xscmd command

You can use the **`xscmd`** command from the server side to show the map size as described in "Starting WebSphere eXtreme Scale servers" on page 116.

A simple performance test

This example does not go into the deeper intricacies of performance testing. It focuses instead on a simple test to determine the response time improvement that is achieved by using WebSphere eXtreme Scale cache. In the simple test scenario and topology for the purposes of this book, the performance with the cache was nearly five times faster than without the cache.

There are many tools to test performance of applications, and one such tool to test response times is WEb INspector REmote (*weinre)*, which you can download from the following website:

http://people.apache.org/~pmuellr/weinre/docs/latest/

Weinre is a remote web debugger where you can measure response times directly. It is especially helpful for testing mobile applications and can also be used for remote debugging on real devices.

When you run weinre on your workstation, it opens a port for communication with a device, which by default is `8081`. In the Worklight application, you must insert one line in the main HTML file in the project. In the CAB application in this scenario, the following line was inserted into `CAB.html`, the main HTML file:

```
<script src='http://yourworkstationIP:8081/target/target-script-min.js#anonymous'>
</script>
```

where *`yourworkstationIP`* is the machine where weinre was installed, which for development-time testing is normally the development machine.

For comprehensive performance testing, you need a full scale performance tuning for both Worklight server and eXtreme Scale, and your mobile application.

3.5 Summary

How to use WebSphere eXtreme Scale in a Worklight mobile application was illustrated by using the following main steps:

1. Prepare the Worklight server for the WebSphere eXtreme Scale run time.
2. Develop the Worklight customization with Java code to interact with WebSphere eXtreme Scale. You can configure the customization so that it can communicate with either WebSphere eXtreme Scale or the DataPower XC10.
3. Start the customization from the Worklight HTTP adapter. The HTTP adapter eventually returns the cache data to the mobile client.

Enterprise caching can run on both distributed platforms and IBM System z (mainframe). Because there are currently applications still running on mainframes, they play a key role for many industries. Consider the following benefits of running the cache on the mainframe:

- Good performance and high availability
- Co-located integration with existing applications on mainframe for the cache
- Better performance and better connectivity
- Linear scaling for WebSphere eXtreme Scale servers, especially when you run all servers on the lightweight Liberty profile
- Full use of mainframe resources, including processor and storage
- Consistent management
- Powerful monitoring capabilities, for collecting fine-grained monitoring data

4

Enterprise caching in an ESB environment

Enterprise caching in an enterprise service bus (ESB) environment is described. Information is provided about the business scenarios where applying caching in the ESB environment is critical. Also described are technical solutions to solve the problem, products that are used, and how the solution is implemented.

The following sections are described:

4.1 Caching with an enterprise service bus

Companies often use an ESB to make data from one system available to other systems. It usually offers support for message transformation, routing, and several transport protocols.

Using cached data from an ESB provides the following benefits:

- Improved response time. Data is returned immediately based on values that are stored in cache without the need to start requests in the target system every time.
- Reduced application load. Many requests to the target system from multiple systems can slow the system or even make it unavailable for business users. By using a cache, requests can be returned immediately without the need to access the target system.

ESB integrations are usually stateless. However, some scenarios require information to be stored between calls to external systems:

- Coordinated request-reply. If the target system responds slowly or asynchronous messages are used, information about how to return the response must be stored externally.
- Aggregation. When calling several systems in parallel, a status of aggregation is usually stored in external storage.

Usually, an ESB uses a database for storing states if there is a requirement for a fault tolerant solution. However, if low response times and high message rates are required, the database can quickly become a bottleneck. This scenario is where using an external cache can help to achieve the required service level agreements (SLAs).

4.1.1 Accessing an external cache from an ESB

WebSphere eXtreme Scale provides programming interfaces to interact with the cache. When you use these interfaces from inside of an ESB, the main difference is the type of transport that is used:

- Java ORB. This type of transport can be used by IBM WebSphere Message Broker or IBM WebSphere Enterprise Service Bus, which supports Java. This transport has several advantages, such as tighter integration with the product, near-cache with custom eviction, and better performance.
- HTTP. This type of transport can be used by ESBs that do not support Java, such as IBM WebSphere DataPower SOA Appliances.

Because there are differences in the way applications use an application programming interface (API), depending on the transport, the following sections provide examples for WebSphere DataPower by using HTTP and WebSphere Message Broker using Java ORB.

4.2 DataPower XC10 as a cache solution for DataPower XI52

This section describes enterprise caching in an ESB environment where a DataPower XI52 appliance is used as an ESB. A representative business scenario is provided from a telecom company where applying side caching in their ESB environment adds business value.

4.2.1 Introduction to the business scenario

A large telecom company with millions of subscribers defined strict SLAs. The retail shops and third-party companies that access the portal system must not experience low response times, and waiting for an operation is unacceptable. The portal provides many services that have high throughputs.

Mobile Station Integrated Services Digital Number (MSISDN) and subscription type data is used in many business critical processes such as order management and added value services. This piece of client information is obtained from various back-end systems such as billing, a customer relationship management (CRM) system, and a customer profile database, either through a web service or directly from a database. Getting this piece of client information takes no longer than a few milliseconds to meet the SLAs.

4.2.2 Solution overview

A web service call is made to the DataPower XI52 Integration Appliance to obtain prepaid or postpaid information. If the web service response is already in the side cache (DataPower XC10 Caching Appliance), the integration appliance (DataPower XI52) returns that result. Otherwise, the DataPower XI52 calls the IBM DB2® database followed by an external web service call to a back-end system. The results are then placed in the DataPower XC10 as a cached response for future use by subsequent calls.

This solution includes the following benefits:

- Improved response time, which decreases the number of calls to slower systems (such as databases), and at the same time, protects critical back-end servers from overloading
- Caching efficiency by increasing the number of back-end servers that are involved in processing logic while generating aggregated responses
- Reduced costs by caching data from external paid services and calling those paid services less frequently

Figure 4-1 shows an overview of this solution. The DataPower XI52 Integration Appliance is shown in the integration layer along with the DataPower XC10 Caching Appliance, which is used as a side cache. The use of the side cache reduces the load on the back-end systems, such as the Customer Database, Billing, and CRM systems, by caching information that was previously retrieved from those back-end systems.

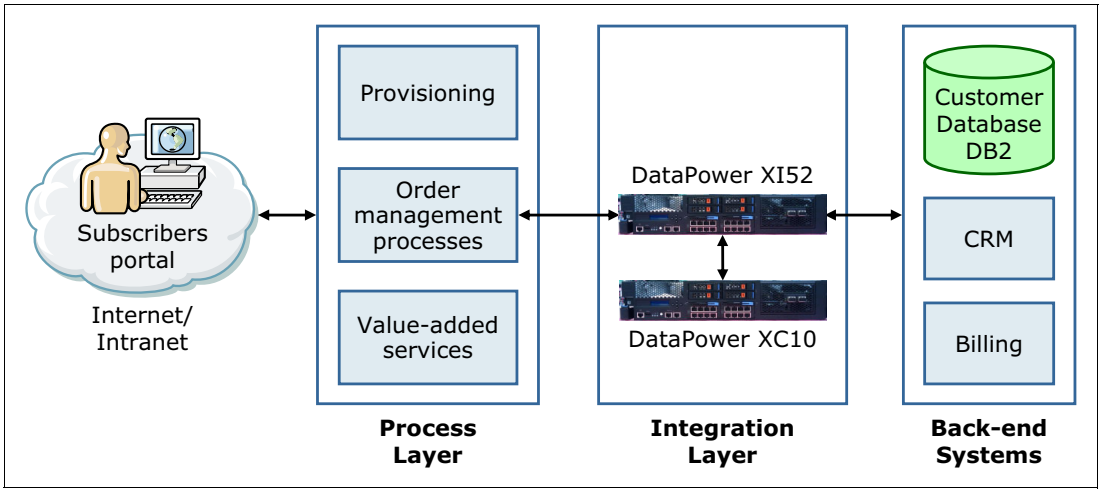

Figure 4-1 Solution scenario overview

4.2.3 Technical implementation

This section describes the configuration of an XML Firewall Service on DataPower XI52 to show the client information service. This service gets required client data for a transaction through web service calls to the back-end system (billing and CRM) and to the customer profile database. The responses from the two service calls are aggregated and cached in the DataPower XC10 appliance. This aggregated information is cached in a dedicated grid that is accessible with a unique key, in this case, the customer telephone number (CTN).

To complete the scenario, follow these steps:

1. Create a simple data grid on the DataPower XC10 Caching Appliance
2. Configure the WebSphere DataPower XI52 Integration Appliance
3. Create an XML Firewall service
4. Create a Web Service Proxy service

Creating a simple data grid on the DataPower XC10

To create a simple data grid, follow the instructions in 6.3.2, “Creating a data grid on the WebSphere DataPower XC10” on page 119. This simple data grid is the side cache for the DataPower XI52 Integration Appliance.

Configuring the WebSphere DataPower XI52

To configure the DataPower XI52 integration appliance to work with the DataPower XC10, you must create the following services:

- XML firewall service. This service is used to communicate with the DataPower XC10, a back-end server, and the database. It acts as a loopback proxy, which is needed because there are two back-end systems instead of just one.
- Web Service Proxy service. This service is used to display the DataPower ESB logic as a web service to external clients. The endpoint acts as an XML firewall service.

Creating the XML configuration files

To communicate with the caching appliance, the DataPower XI52 uses the REST V2 API provided in the following set of configuration files:

- `cache-config.xml`

  This file defines parameters to access the DataPower XC10, its host name, grid, credentials, and map structure.

  Example 4-1 shows the `cache-config.xml` file that is used in this solution.

  Example 4-1 The cache-config.xml file

```
<cache url="172.17.218.198" timeout="30" resource="resources/datacaches"
grid="sw215grid" username="myuser" password="passw0rd" >
    <entry type="GetClientInformation" ttl="60"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cli="http://sw215/clientinformation">
      <key label="ctn"
xpath="/soap:Envelope/soap:Body/cli:GetClientInformation/cli:ctn" />
   </entry>
</cache>
```

  This file uses the following elements:

  – The `<entry>` element

A `cache-config.xml` file can contain multiple `<entry>` elements. Each `<entry>` is meant to represent a unique message format, for example, a request for `getCustomerAddress` as opposed to a request for `getAccountBalance`. By default, the `type` parameter is checked against either the local name of the root element of the XML or the local name of the root element of the SOAP Body. The type of checking depends on the version of `cache-get.xsl` that is used.

To match on another message format identifier, add the following line in the `cache-get.xsl` file:

```
<xsl:variable name="type" select="local-name(/*[1])" />
```

– The `<key>` element

You can define multiple `<key>` elements within each `<entry>` element. The idea is that, in most cases, a single data element in the input message is not enough to uniquely identify a cache entry for the entire request message. Thus, multiple data elements can be extracted and concatenated to make a complete, unique key. For each `<key>` element, use the **`label`** parameter to define a static name and the **`xpath`** parameter to specify where from the input message to extract the actual data.

Example 4-2 looks at a web service request format to understand how the data is mapped.

Example 4-2 Incoming web service sample request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cli="http://sw215/clientinformation">
   <soapenv:Header/>
   <soapenv:Body>
      <cli:GetClientInformation>
         <cli:ctn>1647345811</cli:ctn>
         <cli:requestID>1</cli:requestID>
      </cli:GetClientInformation>
   </soapenv:Body>
</soapenv:Envelope>
```

- `cache-get.xsl` (referring `cache-util.xsl`)

  This file is used in processing logic to get cached reply.

- `cache-set.xsl` (also referring `cache-util.xsl`)

  This file is used in processing logic to save aggregated web-service response.

Integrating WebSphere DataPower XC10 and XI50 Appliances: These XML and XSL files can be downloaded as part of the code sample in the Integrating WebSphere DataPower XC10 and XI52 Appliances article on the IBM developerWorks website:

http://www.ibm.com/developerworks/websphere/library/techarticles/1111_nijhawan/1111_nijhawan.html

Uploading files

The processing policy uses XML configuration files and templates and other artifacts. Upload the files to the DataPower XI52 Integration Appliance by following these steps:

1. Log in to your XI52 appliance.
2. On the **Control Panel**, select **File Management** under Files and Administration, as shown in Figure 4-2 on page 50.

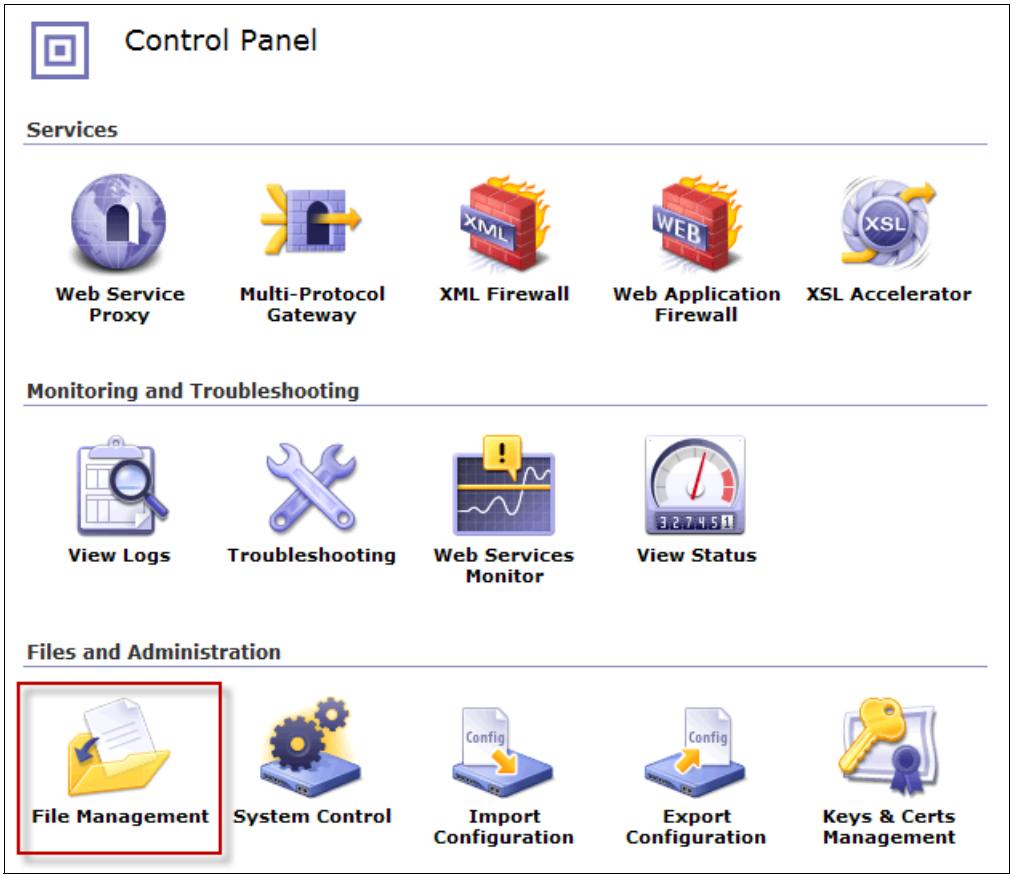

Figure 4-2 DataPower XI52 Control Panel options

3. On the File Management panel, find the folder named `local:` and select **Actions** → **Upload Files** next to the folder, as shown in Figure 4-3.

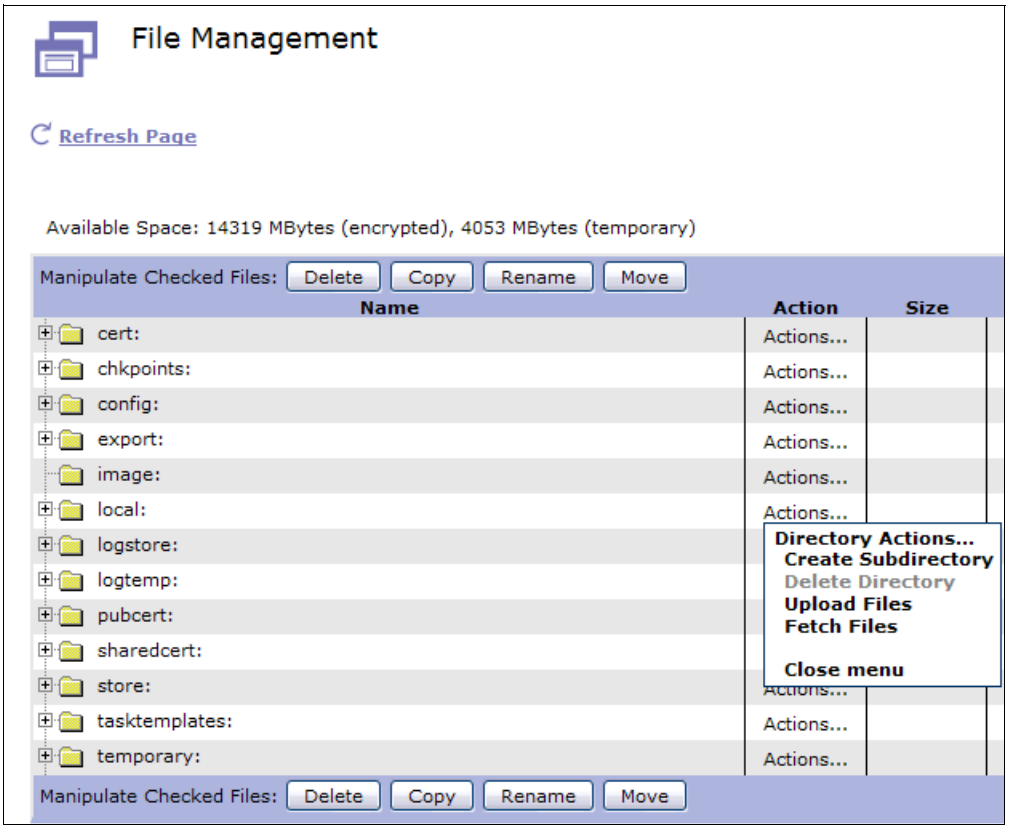

Figure 4-3 File management console to upload files to local

4. You need the following files for this scenario, which are required to process the request for client information. The contents of these files is provided in “Additional files for Chapter 4”. Create these files in a location accessible to the XI52 and save them with the contents listed in the Appendix.

   – `CLIENTINFO.wsdl`
   – `ClientInfo.xsd`
   – `ClientInfoBE.wsdl`
   – `cache-config.xml`
   – `cache-get.xsl`
   – `cache-set.xsl`
   – `cache-util.xsl`
   – `backend-stuff.xsl`

5. Select the files, as shown in Figure 4-4 and then click **Upload**.

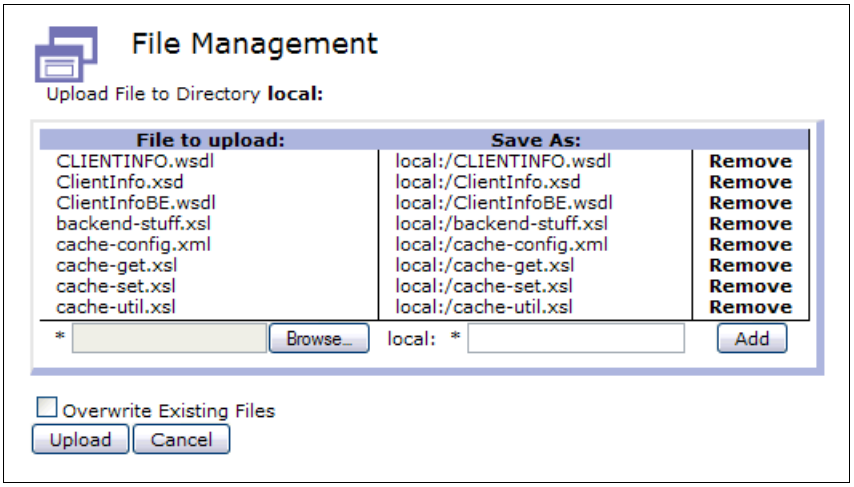

Figure 4-4 Uploading files to the appliance

6. When the files are uploaded successfully, click **Continue.**
7. Click **Control Panel** in the upper-left corner to return to the control panel.

Creating an XML Firewall service

In this scenario, an XML Firewall service in the XI52 is used to do the following functions:

- Access the back-end servers, including the web services-enabled server and database server
- Aggregate the multiple responses into a single response

The XML Firewall is designed to process generic XML requests and responses that are transmitted over HTTP or HTTPS. The XML Firewall uses a single protocol and contains one processing policy with a set of request, response, two-way, and error rules. Its configuration defines the listening IP address-port pair and general threat protection. Although the design of the XML Firewall is to process XML documents of all types, including SOAP-formatted messages, it can accept unprocessed (text/binary) documents as well. Through the processing policy, the XML Firewall can apply all of the various processing actions to the request and response message, regardless of format. Processing can include AAA (Authentication, Authorization, Audit), transformations, schema validation, logging, and cryptographic operations. Like all other DataPower services, the XML Firewall can be configured to proxy remote services. However, one of the commonly used features of the XML Firewall is to define the configuration as a loopback service. In this scenario, the loopback service is used so that a back-end service does not have to be defined.

To create an XML Firewall service:

1. From the **Control Panel**, select **XML Firewall,** as shown in Figure 4-5.

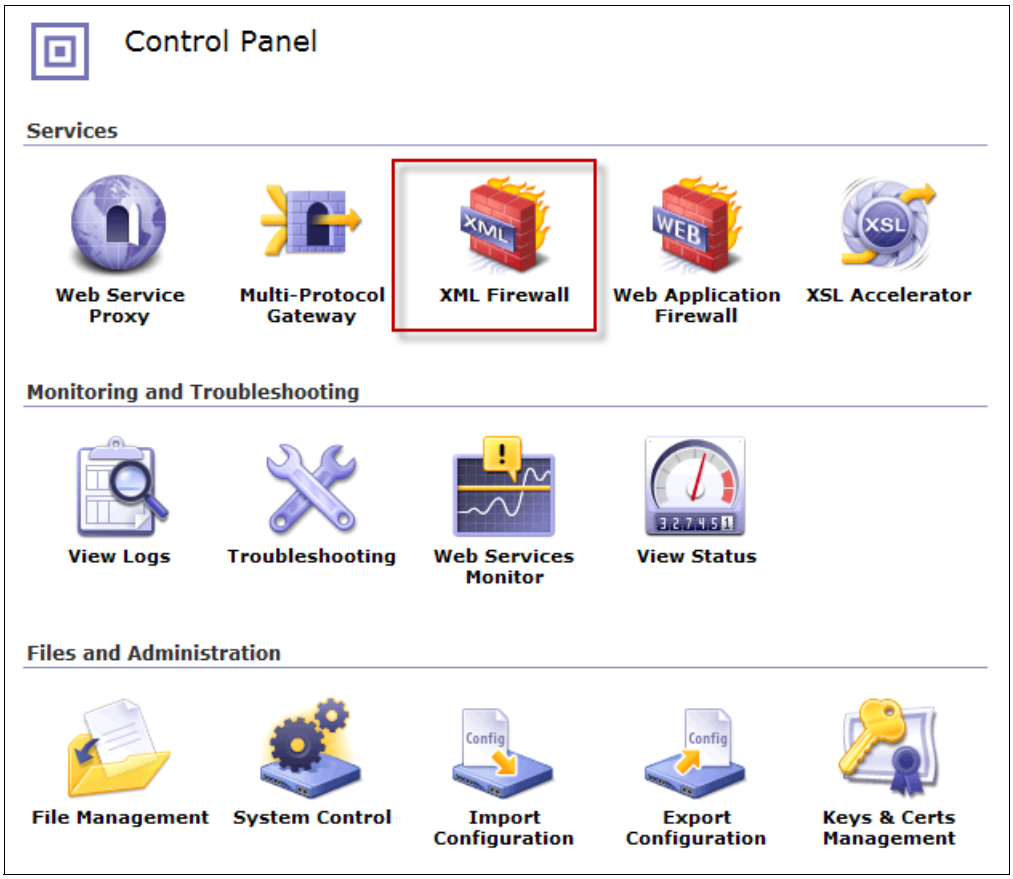

Figure 4-5 Starting with XML Firewall configuration

2. Click **Add Advanced**, as shown in Figure 4-6.

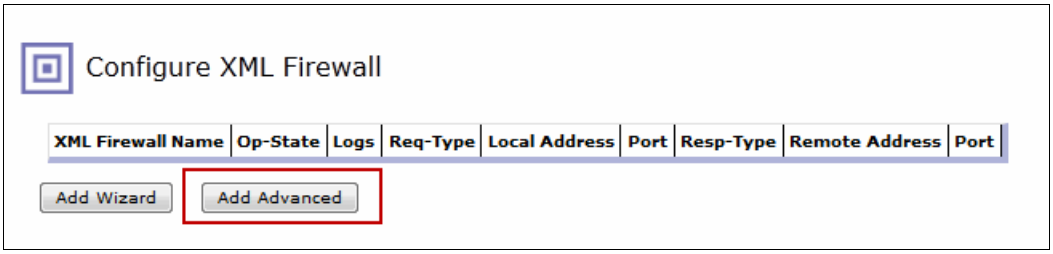

Figure 4-6 Advanced XML Firewall configuration

3. The General Configuration tab is displayed. Specify the Firewall Name **SideCache_XMLFW**, set the Firewall Type to **Loopback**, and set the Port Number to **8001**, as shown in Figure 4-7.

> **Default local IP address:** The local IP address is set to `0.0.0.0` by default, which means that the service accepts connections from all network interfaces on the appliance.

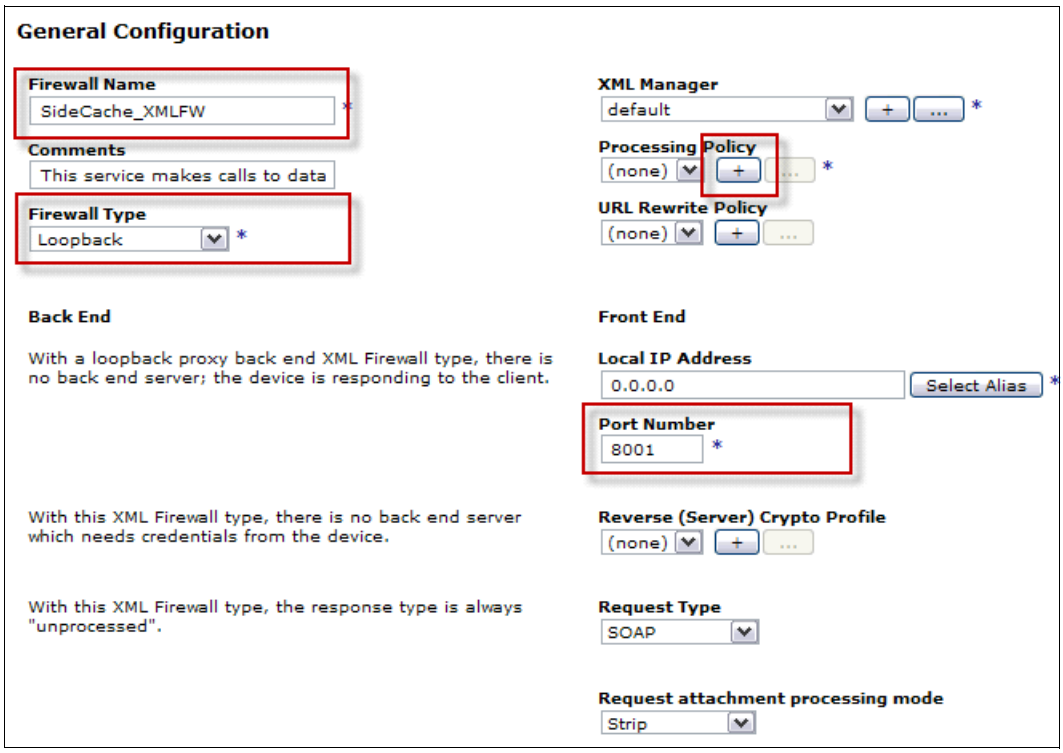

Figure 4-7 Setting general attributes of XML Firewall service

4. Create the XML Firewall Processing Policy by pressing the plus sign (+) next to the drop-down menu for Processing Policy. The Configure XML Firewall Style Policy window is then displayed, as shown in Figure 4-8.

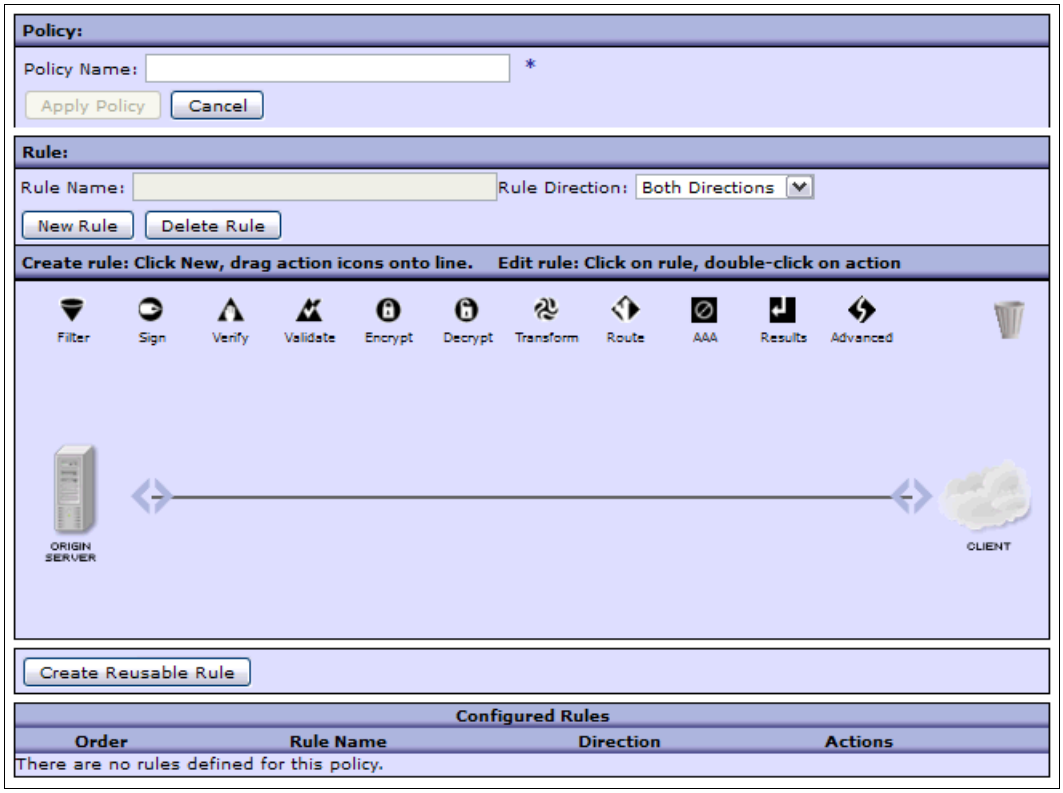

Figure 4-8 Configure XML Firewall Style Policy window

5. Set the Policy Name to **SideCache_XMLFW_Policy**, set the Rule Direction to **Client to Server**, and then create a rule by clicking **New Rule**, as shown in Figure 4-9.

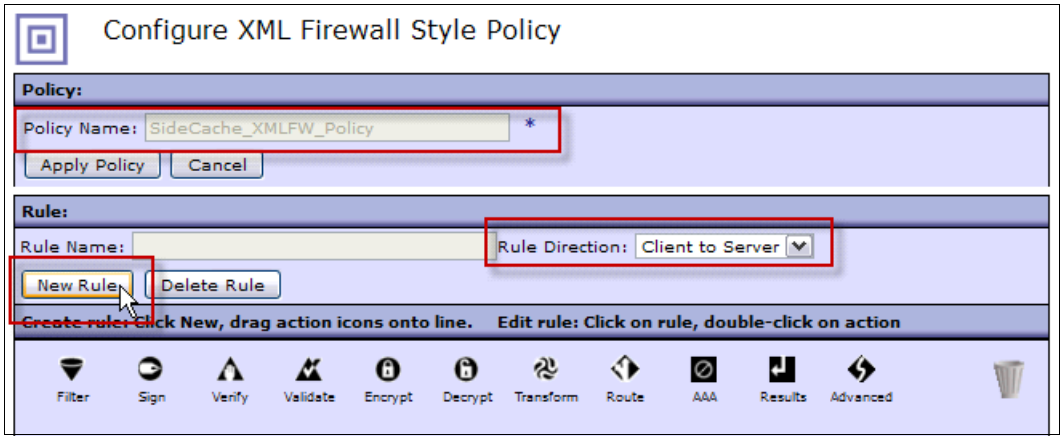

Figure 4-9 Configure Policy: create new rule

6. The *matching rule* is represented by the icon in the middle of the line with the yellow box around it, as shown in Figure 4-10.

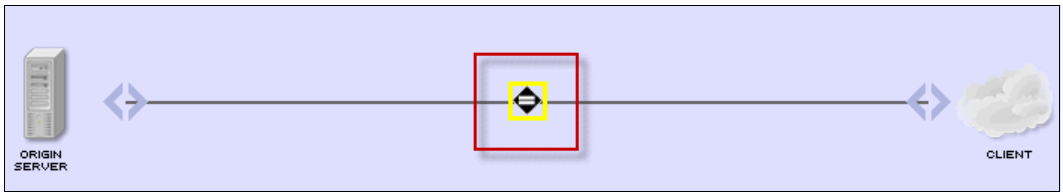

Figure 4-10 Configuring the matching rule

7. Double-click the **Matching Rule** icon to display the Configure a Match Action window, which is shown in Figure 4-11. Press the plus sign (+) next to the Matching Rule field to create a rule.

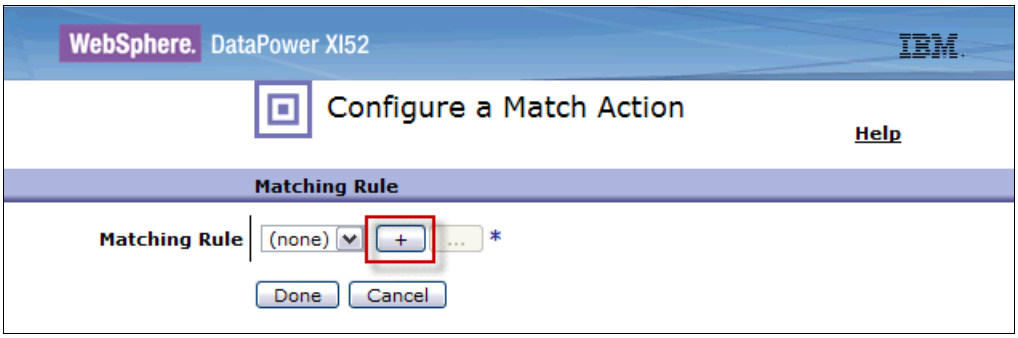

Figure 4-11 Configure a Match Action window

8. In the **Main** tab, specify the name **match_all_rule** in the Name field, leaving the rest of the attributes set at their default values, as shown in Figure 4-12. Then, switch to the **Matching Rule** tab at the top of the page to define the matching rule.

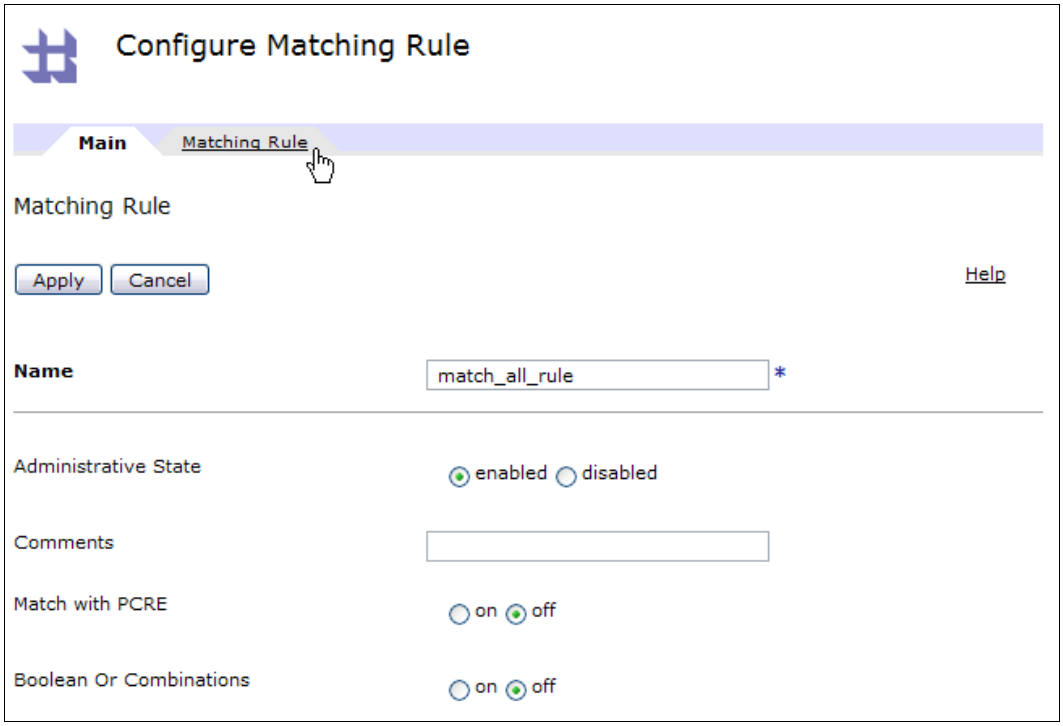

Figure 4-12 Configure Matching Rule window

9. Create a rule by clicking **Add**, as shown in Figure 4-13.

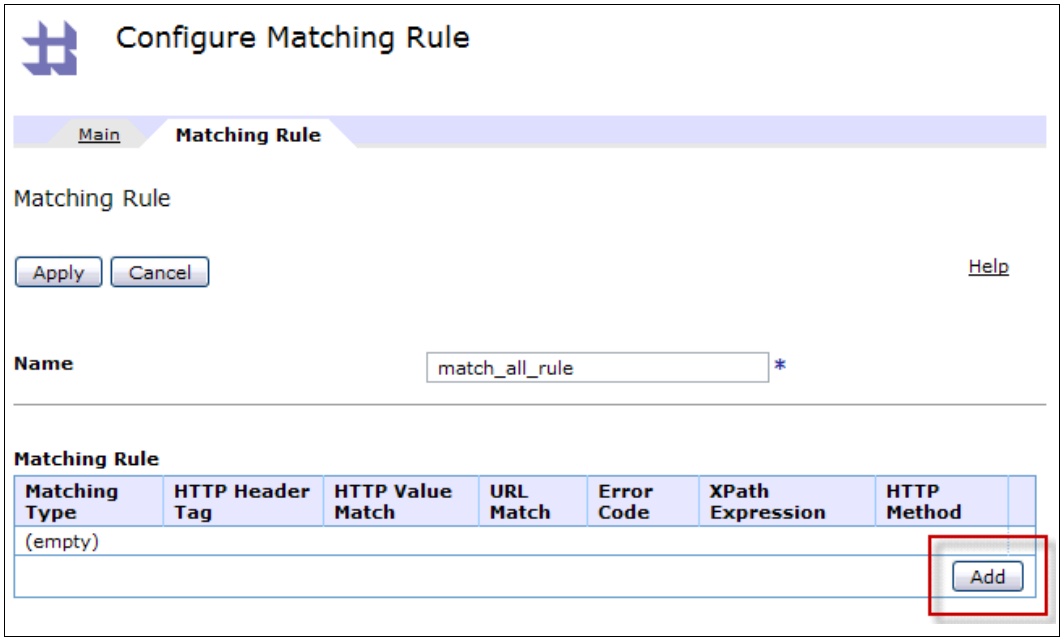

Figure 4-13 Configure Matching Rule window

10. Set the Matching Type to **URL** and the URL Match value to an **asterisk** (*), as shown in Figure 4-14. This selection indicates that the service accepts all requests. Click **Apply**.

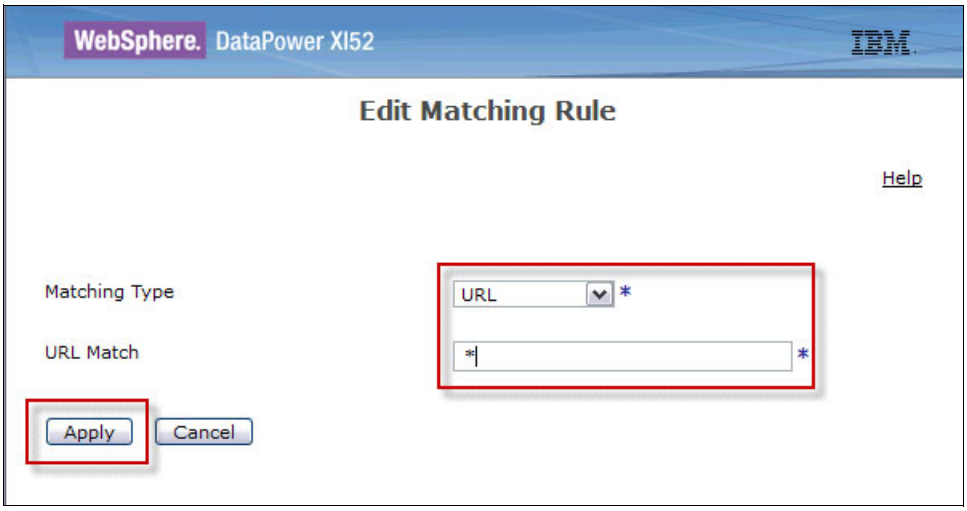

Figure 4-14 Setting Matching Rule to accept all requests

11. The new rule is now shown on the previous window. Click **Apply** and then click **Done**.
12. The Configure XML Firewall Style Policy window is displayed again. The yellow box is no longer visible around the matching rule icon. Next, define a Transform Action to call `cache-get.xsl`. Locate the **Transform** action from the Action toolbar and drag it onto the line to the right of the match rule icon, as shown in Figure 4-15.

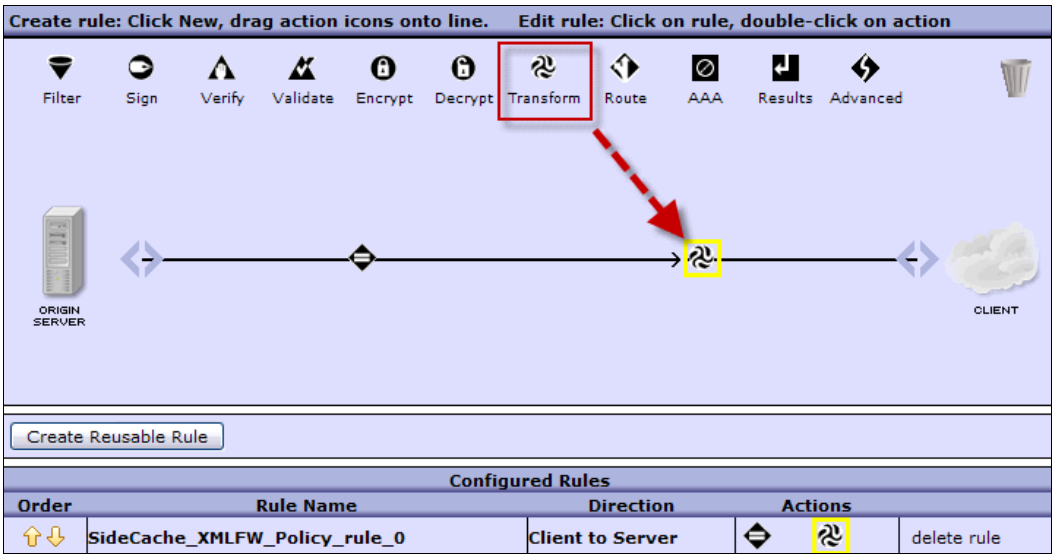

Figure 4-15 Adding cache-get.xsl transform action

13. Double-click the action to display the **Configure Transform Action** window, as shown in Figure 4-16. Click the drop-down menu next to the Input field and select **Input** from it, which then puts the word **INPUT** into the text field. The context of INPUT means that the transform action takes the original client request. Ensure that **local:///** is in the first drop-down menu beside the XSL style sheet label and then select the file **cache-get.xsl** from the drop-down menu located below `local:///`. Enter **ToRespondWith** into the text field for the Output context. This context is used in further actions.

> **Context's content:** The `cache-get.xsl` template tries to get a cached response from XC10. If successful, the `ToRespondWith` context holds cached data. If not, the context holds the original client request. Thus, the next processing action depends upon that context's content.

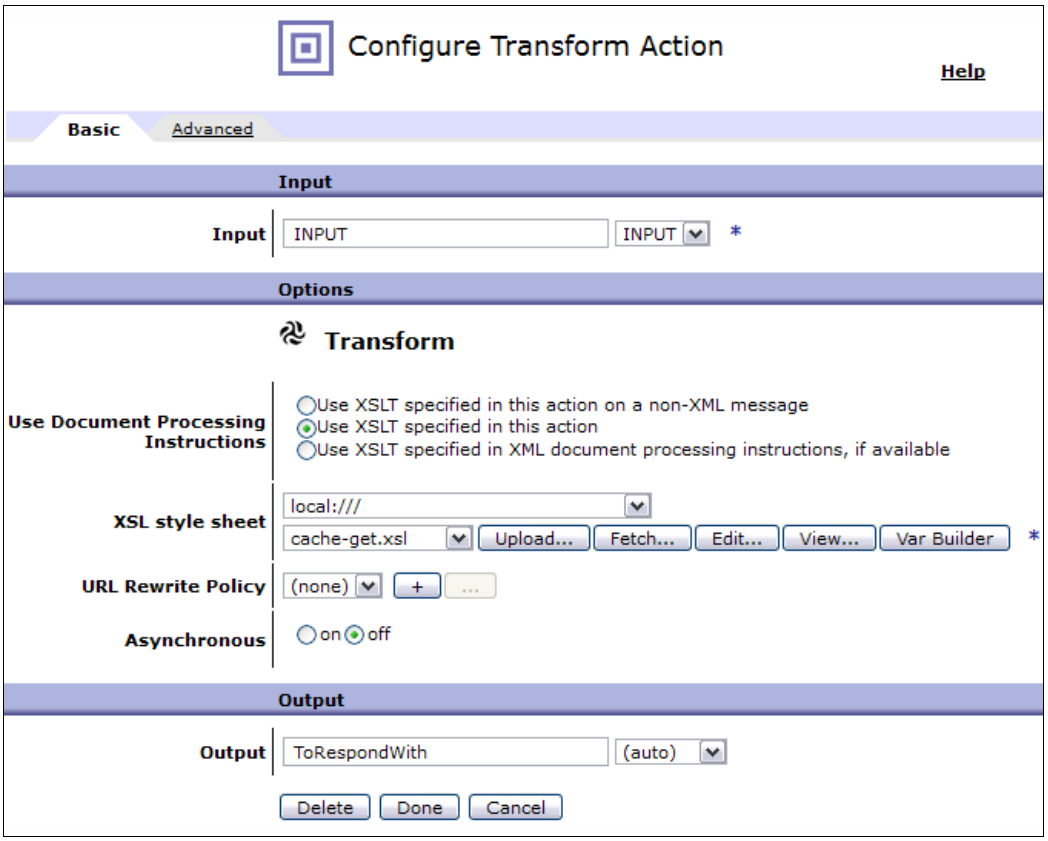

Figure 4-16 Configuring cache-get.xsl transform action

14. Click **Done** to return to the policy definition. Drag the **Advanced** icon to the line to the right of the Transform icon and then double-click to configure the Advanced action. When prompted for the type of the action, select **Conditional** from the list, and then click **Next.** The Configure Conditional Action window is displayed, as displayed in Figure 4-17.

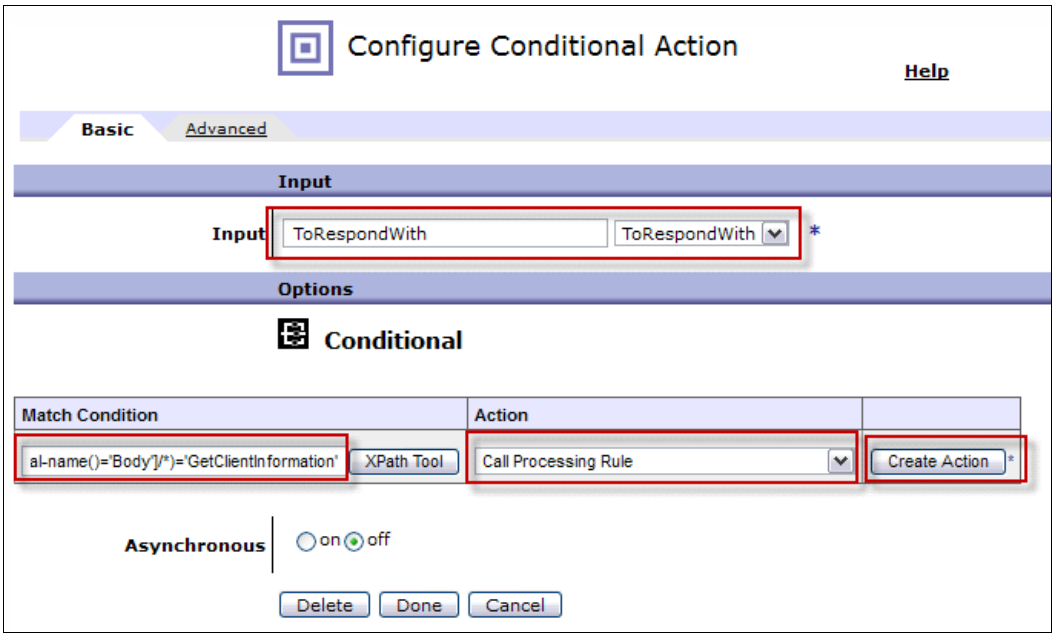

Figure 4-17 Configuring the conditional action

15. For the input, select **ToRespondWith** from the drop-down list to the right of the Input text field, which then completes the text field with ToRespondWith as well. Set the Match Condition to the following text, which assigns the action only in cases where the XC10 has no cached response yet:

```
local-name(/*[local-name()='Envelope']/*[local-name()='Body']/*)='GetClientInfo
rmation'
```

16. Select Call Processing Rule from the Action drop-down list and click **Create Action**, as shown in Figure 4-17. This selection assigns an action to this matching condition.

17. For not-cached requests, start a processing rule to call the targeted back-end systems, aggregate the results, and store those results in the DataPower XC10 Caching Appliance. Set Input context to **INPUT**, set Output to **dp_var1**, and then create a Processing Rule by clicking the **plus sign** (**+**), as shown in Figure 4-18.

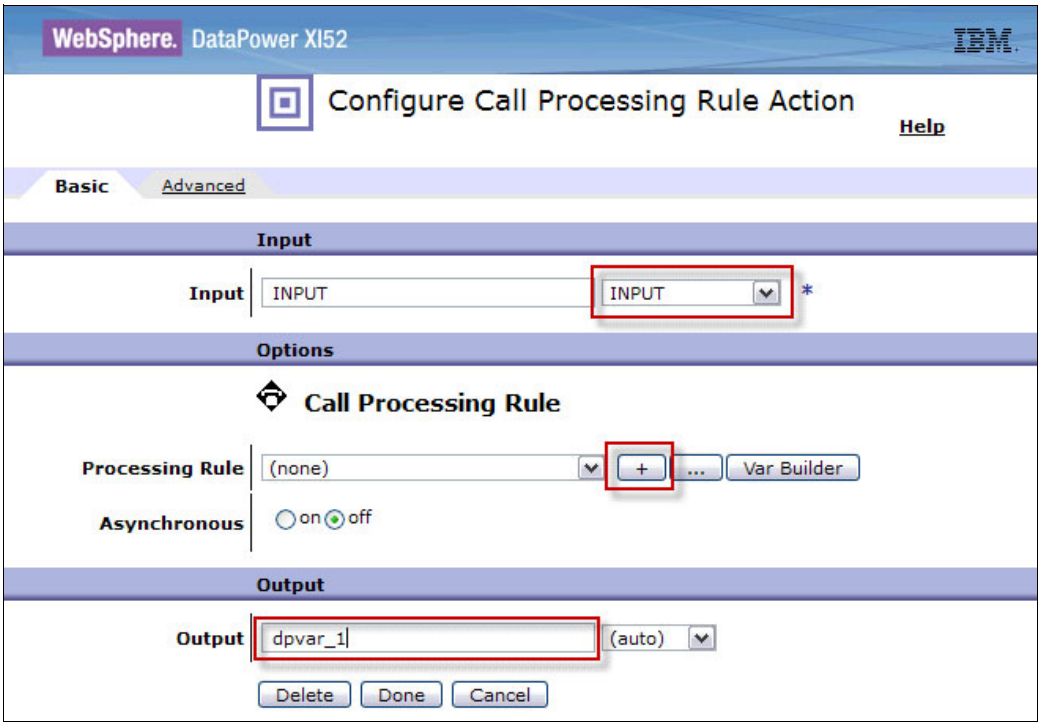

Figure 4-18 Configure Call Processing Rule Action window

18. Clicking the plus sign displays the Configure Processing Rule window. Name the processing rule **SideCache_XMLFW_Call_And_Cache** and set the Rule Direction to **Client to Server**, as shown in Figure 4-19.

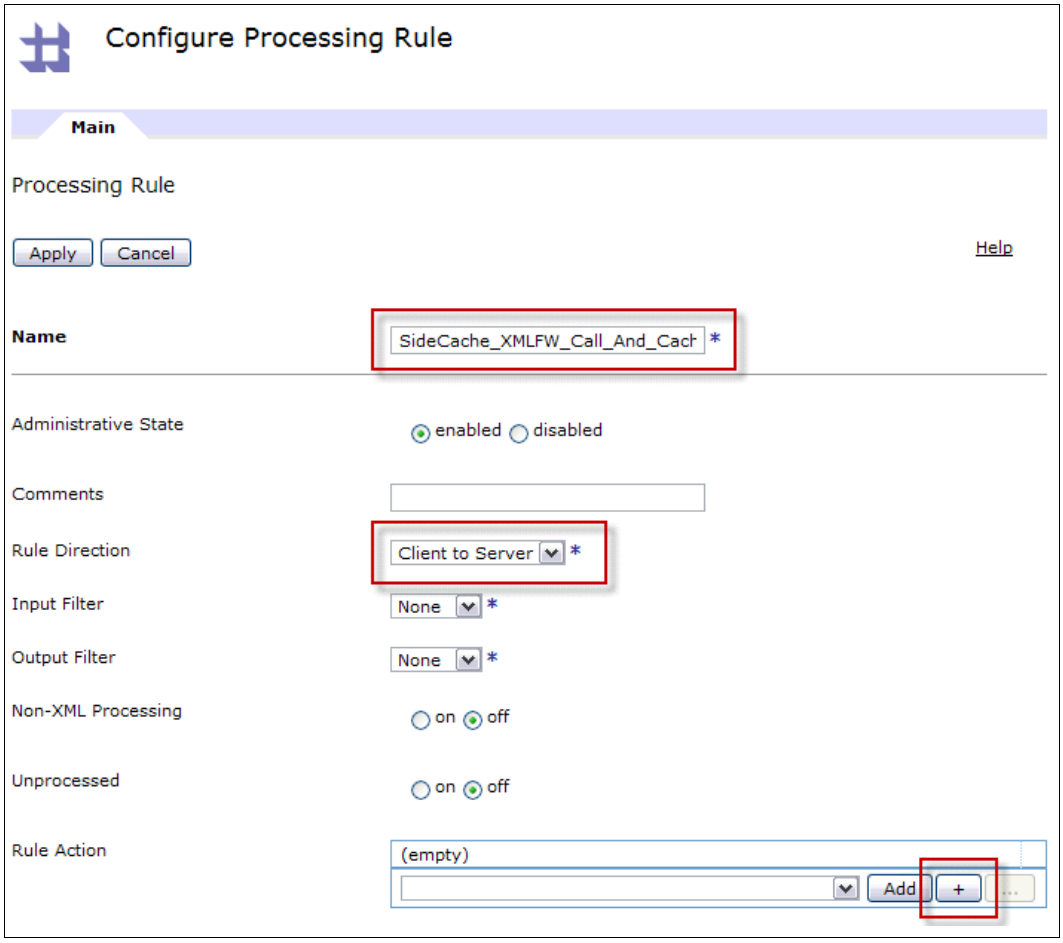

Figure 4-19 Configure Processing Rule actions

19. The processing rule contains two transform actions. Define these actions by clicking the **plus sign** (**+**) next to the Rule Action list. This selection displays a new Configure Processing Rule window. Create the first action, **Call_Backends**, with the information that is shown in Figure 4-20.

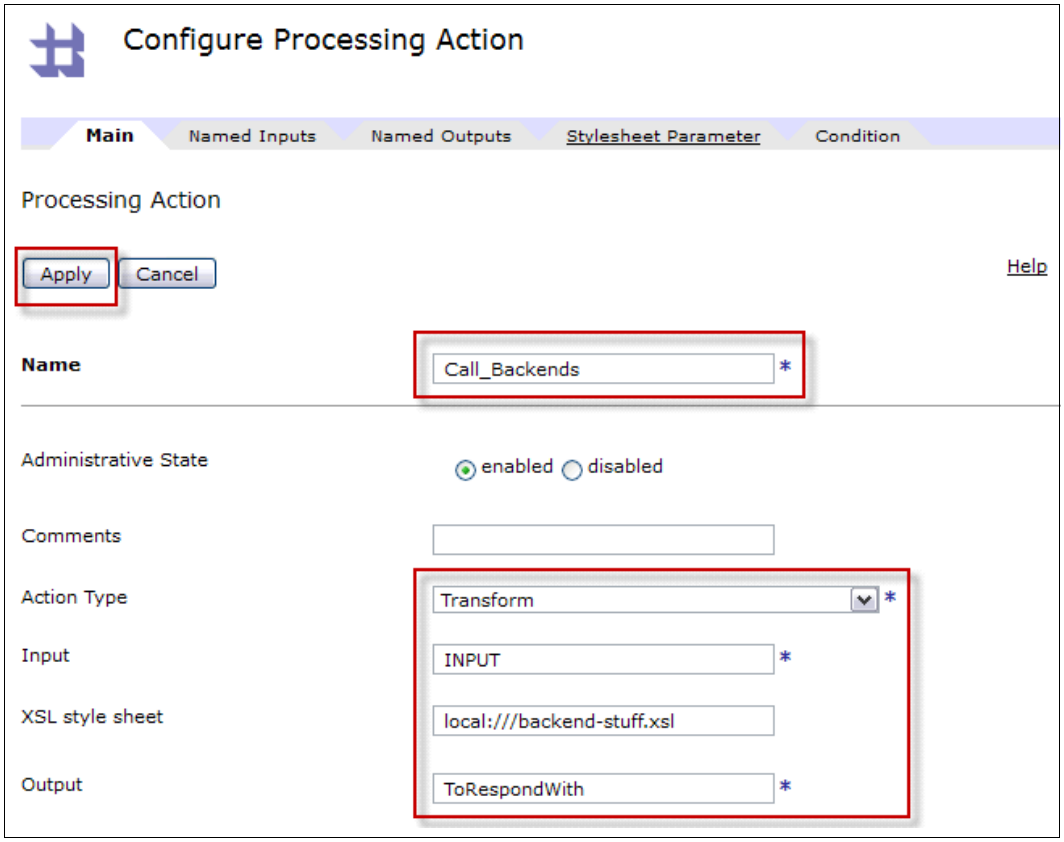

Figure 4-20 Call_Backends rule configuration

Appendix , “backend-stuff.xsl” on page 146 shows the contents of the `backend-stuff.xsl` file, which is used to call back-end systems and to aggregate the results. This file is the XSL file that is defined in the Transform field in Figure 4-20.

20. Click **Apply** to save the rule and the Configure Processing Rule window is redisplayed with the `Call_Backends` action listed. Click the **plus sign** (**+**) again to create the second rule, the **Set_Cache** processing action. The values to set for this action are shown in Figure 4-21.

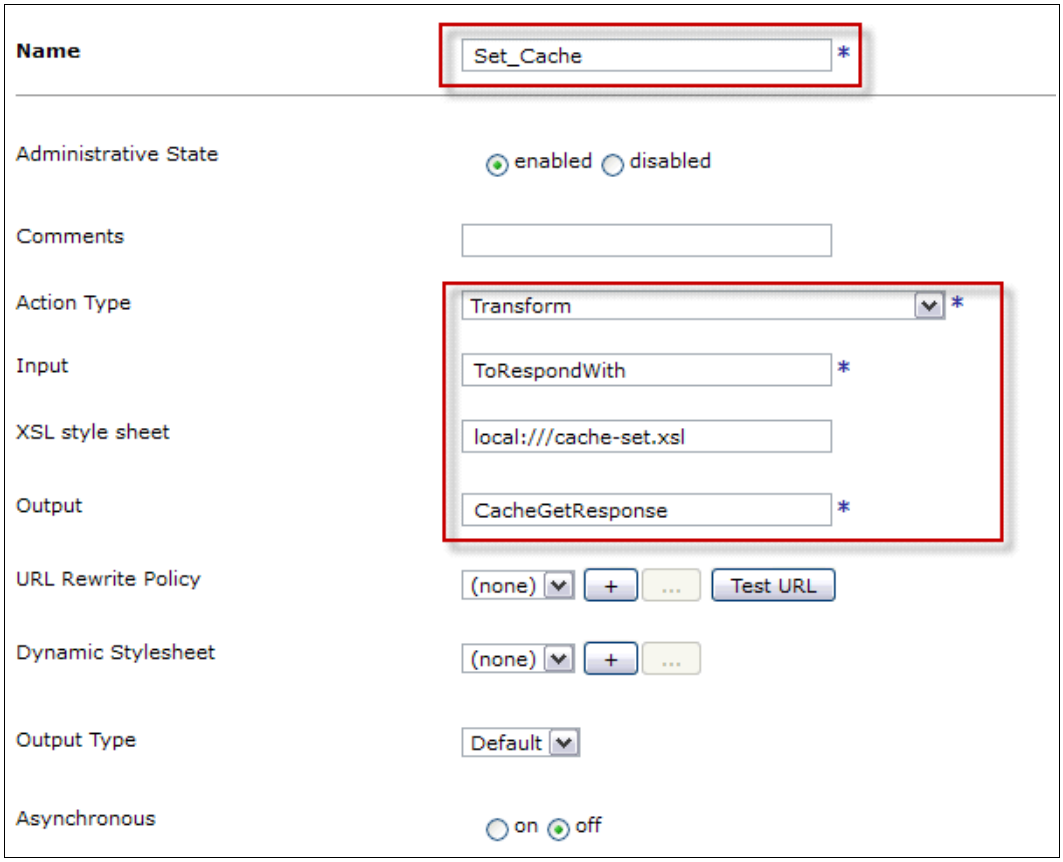

Figure 4-21 Set_Cache action configuration

21. Click **Apply** and the Configure Processing Rule window is redisplayed with the `Set_Cache` action listed.
22. Click **Apply** to save the Processing Rule and return to the Configuring Call Processing Rule Action window.
23. Click **Done** here to return to the Configure Conditional Action window. See your action, `SideCache_XMLFW_Policy_rule_0_condition_0_refaction_call_0` next to the Match Action.

24. Click **Done** to save and display the Configure XML Firewall Style Policy window, this time with the Conditional Action configured. See Figure 4-22.

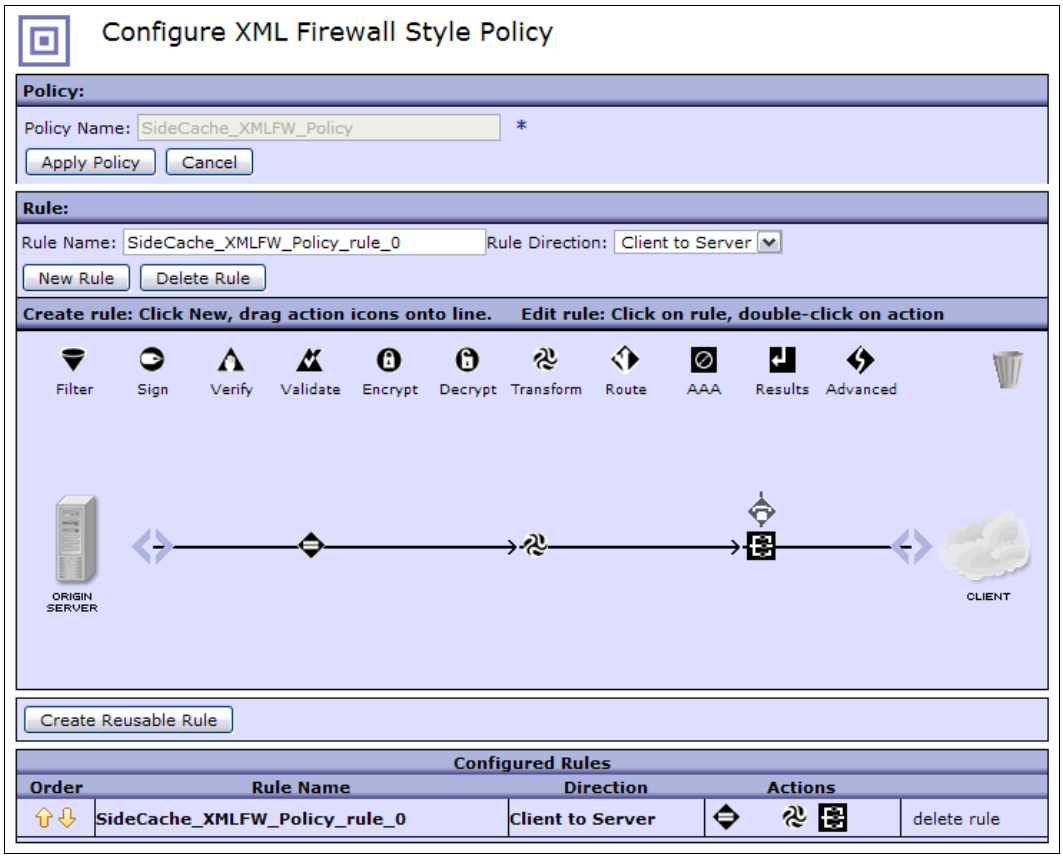

Figure 4-22 Configure XML Firewall Style Policy with Conditional Rule window

25. Add the Results action by dragging the **Results** action icon from the pallet of actions to the line to the right of the Advanced/Conditional action, as shown in Figure 4-23.

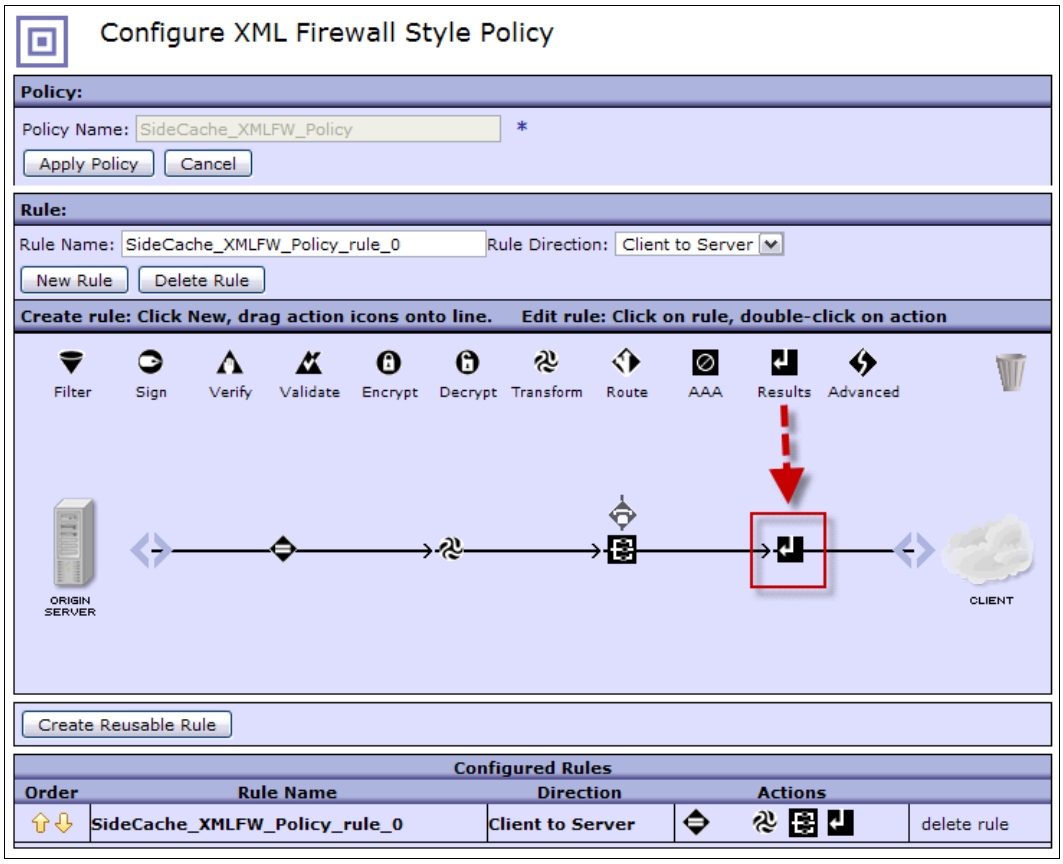

Figure 4-23 Adding the Results action

26. Double-click the icon that you just dropped to configure the Results action. Select **ToRespondWith** from the Input drop-down list and **OUTPUT** from the Output drop-down list, as shown in Figure 4-24. Then, click **Done**.

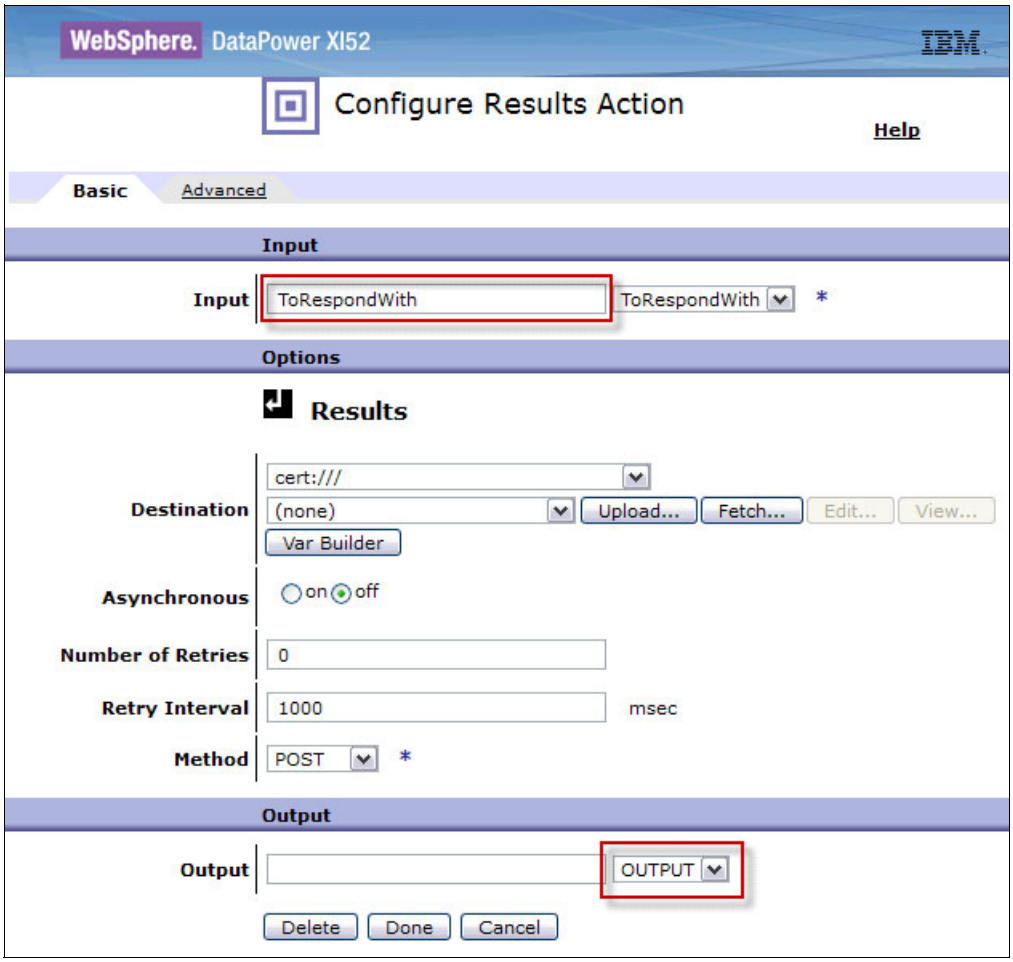

Figure 4-24 Configure Results Action window

27. This selection returns you to the Configure XML Firewall Style Policy window. Click **Apply Policy**.

Creating a Web Service Proxy service

The Web Service Proxy (WS-Proxy) object in the XI52 fully understands Web Services Description Language (WSDL). In fact, you can configure a fully functional WS-Proxy simply by providing it with a WSDL document. When configured, you can configure extra processing requirements and quality of service parameters by using the WebGUI.

To create a WS-Proxy:

1. Log in to your DataPower XI52 Integration Appliance to display the control panel.

2. Click the **Web Service Proxy** icon, as shown in Figure 4-25.

Figure 4-25 Creating a Web Service Proxy

3. Click **Add** to create a Web Service Proxy object.
4. In the **Web Service Proxy Name** field, type `DP_WSP` and click **Create Web Service Proxy**, as shown in Figure 4-26.

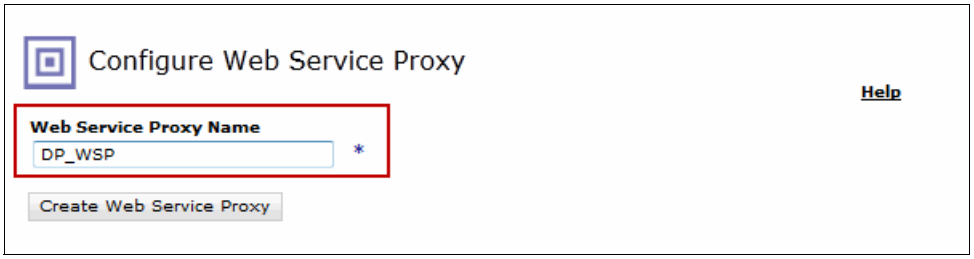

Figure 4-26 Creating a Web Service Proxy

5. The WSDL files tab of the **Configure Web Service Proxy** window opens, as shown in Figure 4-27. You previously uploaded all of the files that are needed for this scenario in on page 49, so select the file **CLIENTINFO.wsdl** from the drop-down list, as shown in Figure 4-27.

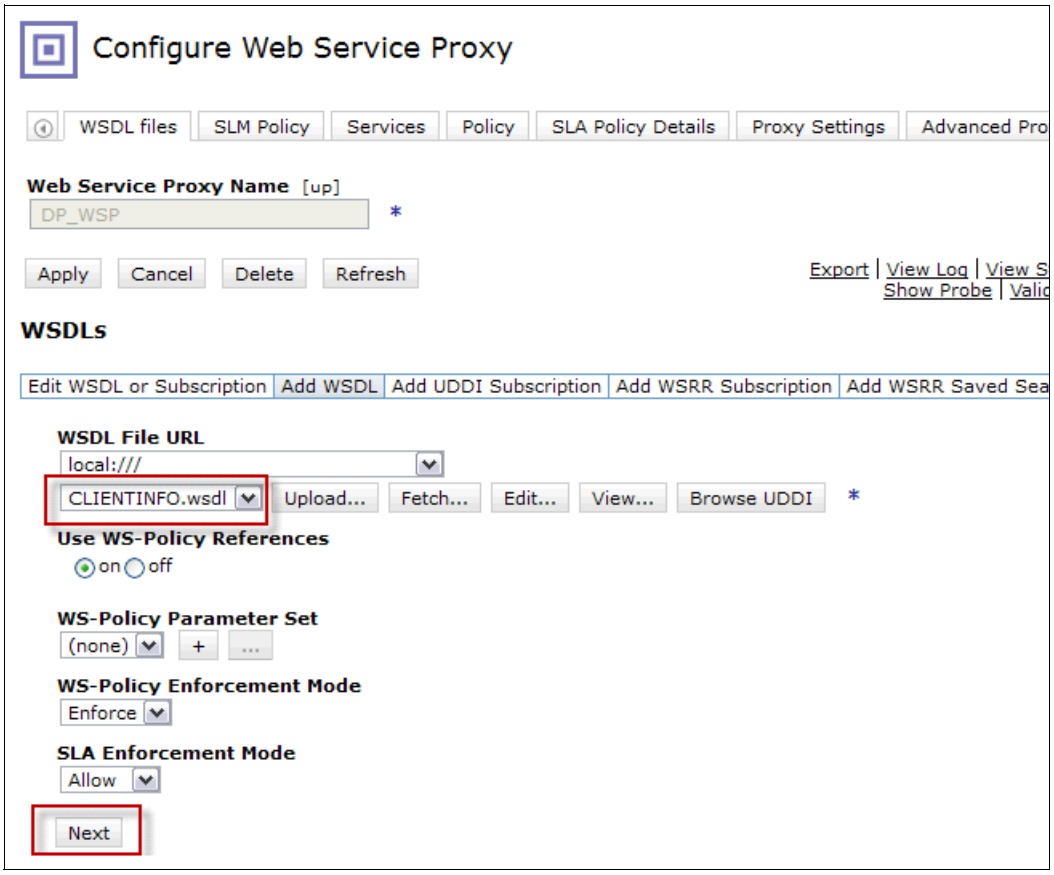

Figure 4-27 Adding a WSDL to a Web Service Proxy

6. Click **Next** at the bottom of the page.

7. Create a front side handler to accept HTTP requests for the WS-Proxy. Use the following procedure to configure the HTTP Front Side Handler:

   a. In the **Local Endpoint Handler** field, click the **plus sign** (**+**) to create an endpoint handler. The list shows the possible front side handlers, as shown in Figure 4-28.

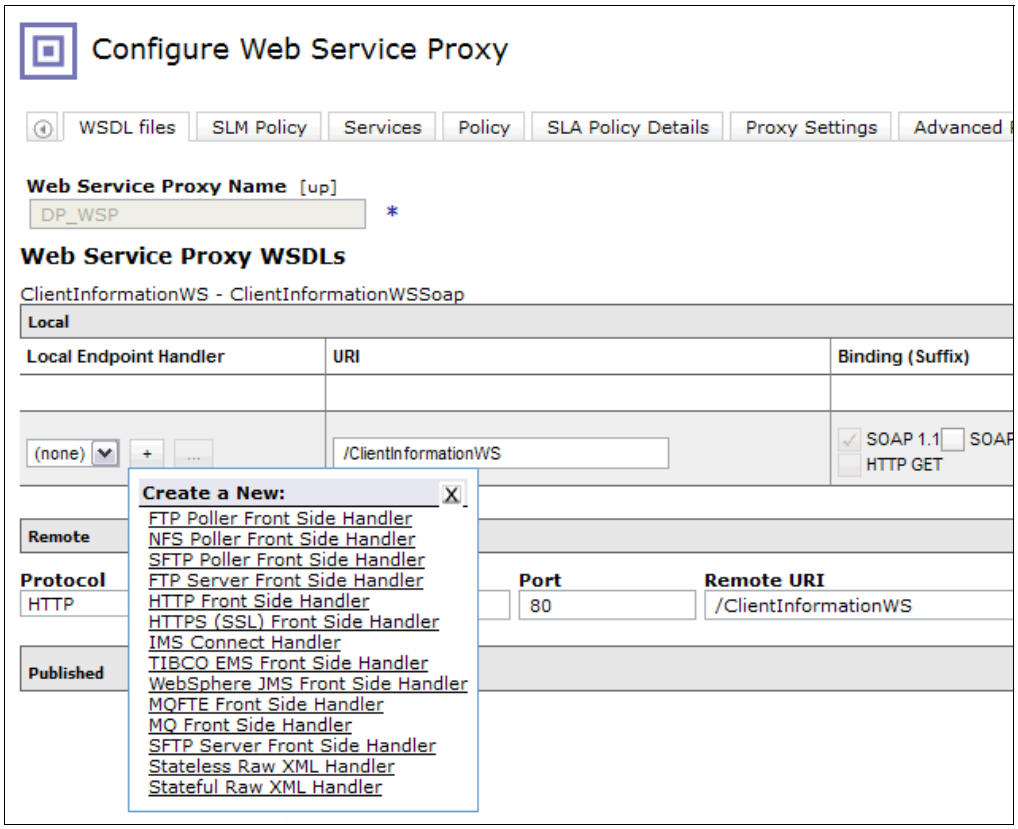

Figure 4-28 Adding a new Local Endpoint Handler

Front side handler: A front side handler is the entry point to several services. A handler listens on a specific IP address port for incoming requests (or polls for messages) and does some level of validation on the requests. Most validation is protocol-specific.

b. Click **HTTP Front Side Handler**. A new window opens that displays the properties of the HTTP Front Side Handler.

c. Configure the new HTTP Front Side Handler with the values shown highlighted in Figure 4-29, leaving the rest of the fields with the default values.

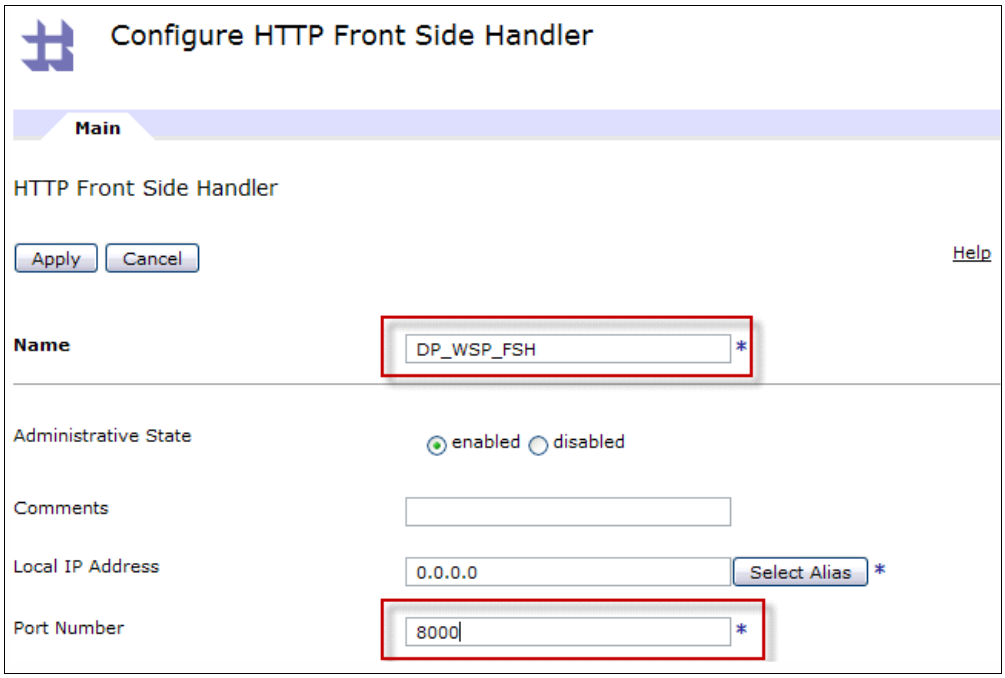

Figure 4-29 Configure the HTTP Front Side Handler

Local IP address: The default IP address is `0.0.0.0`, which indicates that the service is active on all IP addresses that are assigned to DataPower. You can limit the traffic for the WS-Proxy to a specific port on DataPower.

Advanced configuration of the HTTP Front Side Handler: You can filter the incoming requests based on the HTTP protocol level properties. Go to the following address for an explanation of the configuration parameters:

http://publib.boulder.ibm.com/infocenter/wsdatap/v3r8m2/index.jsp?topic=%2Fxi50%2Fhttp-handler_configuring_task.htm

8. Click **Apply** to return to the Configure Web Service Proxy window.
9. Click **Add** next to the Local Endpoint Handler information, as shown in Figure 4-30. This selection saves the handler and moves it up into the row above.

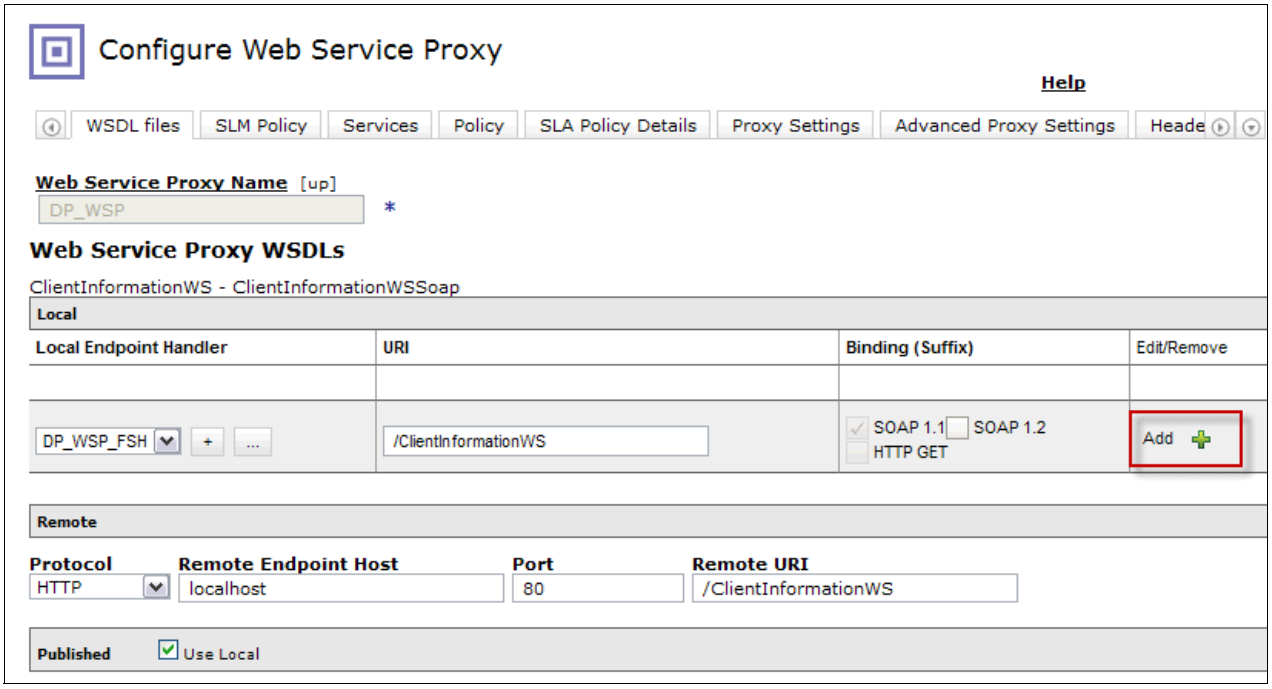

Figure 4-30 Adding local endpoint handler

10. Click **Next** to save the new WS-Proxy service. The Configure Web Service Proxy window is redisplayed and you should see that the service is up and running, as shown in Figure 4-31.

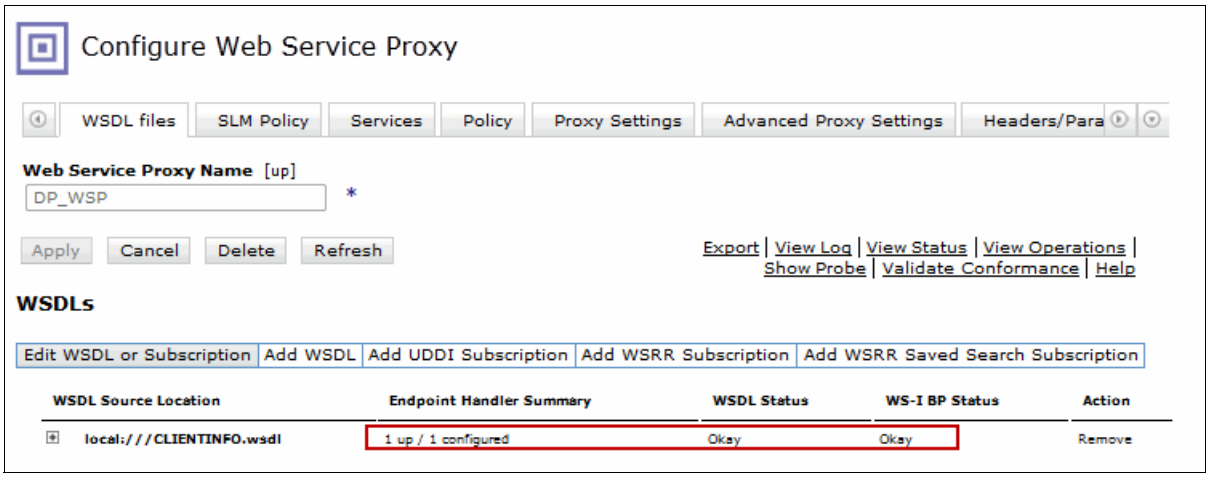

Figure 4-31 WS-Proxy status

4.2.4 Conclusion

These solution benefits are achieved with the XI52 and XC10 configuration:

- A one-hour configuration and implementation
- A simple, drop-in appliance approach (both for the DataPower XI52 and XC10)

- Average IT skills involved (basic XML and XSLT knowledge)

4.3 DataPower XC10: Side cache for WebSphere Message Broker

This section shows how to implement caching solutions when WebSphere Message Broker is used as the ESB. This configuration is slightly different from the scenario in the previous section and uses a different method to connect to the DataPower XC10 Caching Appliance.

4.3.1 Introduction to the business scenario

A railway company has a ticketing system that is available to travel agencies through web services or Representational State Transfer (REST) services and that is available to regular customers by using a web portal or mobile application. The company expanded their railway network, which led to an increase in their client base. However, clients started to complain about the long response times of the ticketing system, indicating up to minutes of waiting times for a simple train schedule inquiry in the social media.

In their current architecture, they are using WebSphere Message Broker as an ESB to help integrate the various back-end systems, such as the train schedule data that is hosted on a DB2 system on z/OS. However, the train ticketing system is struggling to meet the new demands because it was not designed for the current volume of transactions. Scaling the current architecture is an option, but it might prove costly to scale it to meet the new transaction volume.

4.3.2 Solution overview

To solve the problem, a caching solution is added to existing integration flows in WebSphere Message Broker. In the current solution, a web service call is made to WebSphere Message Broker to get the train schedule. A broker transforms the request and uses WebSphere MQ to pass the request to the ticketing system that is running on z/OS. In the proposed solution, if a specific station response is already available in a cache, the response is returned immediately. Otherwise, a message is sent over WebSphere MQ to the ticketing application. The response from the application is transformed from WebSphere MQ to the protocol used by the caller, which is stored in the cache, and returned to the client. Saving whole final responses is a significant performance benefit because there is no need to do extra transformations or aggregation of data from the external system.

4.3.3 Caching options in WebSphere Message Broker

There are several options for caching data from WebSphere Message Broker. Each of them has their own strengths and weaknesses. These options are now described.

WebSphere eXtreme Scale external cache client

WebSphere Message Broker supports custom code in integration flows. Therefore, you can use the WebSphere eXtreme Scale client to access an external cache. This method can help eliminate issues that are related to capacity or losing cached values upon restart. However, a latency penalty might be associated with network connectivity.

You can use either the WebSphere eXtreme Scale grid or the DataPower XC10 appliance. The code that is used to access both types of caching solutions is the same.

WebSphere Message Broker SupportPac IA91

This feature pack allows brokers to access cache data across the flows and supports key-to-value mapping and time-based eviction.

This feature pack is a good option if you are using WebSphere Message Broker V6.1 or V7 and if this level of in-memory caching is sufficient for your needs.

WebSphere Message Broker Global Cache feature

Delivered with the 8.0.0.1 fix pack, this feature embeds WebSphere eXtreme Scale inside WebSphere Message Broker execution groups. Because this cache is embedded, it has imitations that are related to size and the number of objects in a cache. You can use the cache as it is shipped, without modification, and you need to only enable it to be used in flows. In terms of API calls, the main difference between this feature and using the WebSphere eXtreme Scale external client is in getting a reference to a map object. All other calls have the same syntax. Thus, replacing this cache with an external client when needed can be done easily.

It is possible to set up a highly available infrastructure by using more than one broker and built-in replication features. Data is always stored in memory. In addition, it is not possible to set up custom loaders that save it to a database. Currently, it is not possible to override the default configuration of the built-in cache. Thus, if such a modification is needed, an external cache that uses WebSphere eXtreme Scale might be an option.

WebSphere Message Broker Global Cache allows the sharing of data in a broader scope, creating several new scenarios, such as using Global Cache for storing correlation information in a coordinated request-reply pattern. A sample is available in the WebSphere Message Broker Toolkit that uses this concept. Because this information might also be available on another broker or running on a different server, it allows for more flexible horizontal scaling of integration solutions. It also provides better availability because previous correlation information was typically stored in WebSphere MQ queues.

Compared to SupportPac IA91, the WebSphere Message Broker Global Cache allows for better use of memory because data is spread between four containers and is not duplicated in each execution group.

Long-live variables

Long-live variables are a special type of variable in extended SQL (ESQL) code that have a lifetime that is spanned over one execution of a message flow. These variables are easy to use because no extra setup is required. However, keep in mind the following considerations when you use long-live variables:

- Data can be shared only between threads of one flow.
- Support for key value mapping requires looping over a variable that is declared as a ROW type.
- Extra synchronization must be made if several threads are accessing this same variable.
- Data is volatile and must be restored whenever flow is restarted.
- No support exists (unless coded) for time-based evictors.

Conclusion

Selecting a caching solution depends on your requirements. If only a few values are cached (for example, a dictionary of values that are used during message transformation), then an embedded solution, such as the Global Cache feature, should be sufficient. For large sets of data, such as those described in this scenario, an external cache might be a better option. An

external cache gives all available memory to application data and frees up processor cycles to be used for managing the cache. Sometimes benefits from caching whole responses can be seen if values are cached more than 24 hours. Thus, the external cache must be large enough to store all messages.

4.3.4 Technical implementation

In this example, an existing integration flow is modified to benefit from external caching. The target system is showing their services using WebSphere MQ. In WebSphere Message Broker, a request-response Service Facade to WebSphere MQ pattern is used. WebSphere Message Broker patterns allow for quick development of integration flows for common integration problems. This pattern makes WebSphere MQ service accessible using a web service call. An assumption is made that the cache key is available in the response message. If that is not the case, then it should be stored in the request flow. The request flow is restored in the response flow, and used for writing the response to a cache.

Data grid configuration

The data grid is created on an XC10 appliance. For this scenario, a Simple Data Grid template is used that is available with the XC10 appliance. The other two templates, dynamic cache data grid template and session data grid template, are normally used when interacting with WebSphere Application Server.

Request flow

In the request flow that is generated by the Service Facade to WebSphere MQ pattern, a subflow that is called RequestProcessor allows for modifications to the messages before they are sent to the back-end systems. The request flow, which is shown in Figure 4-32, is the only place where modifications take place.

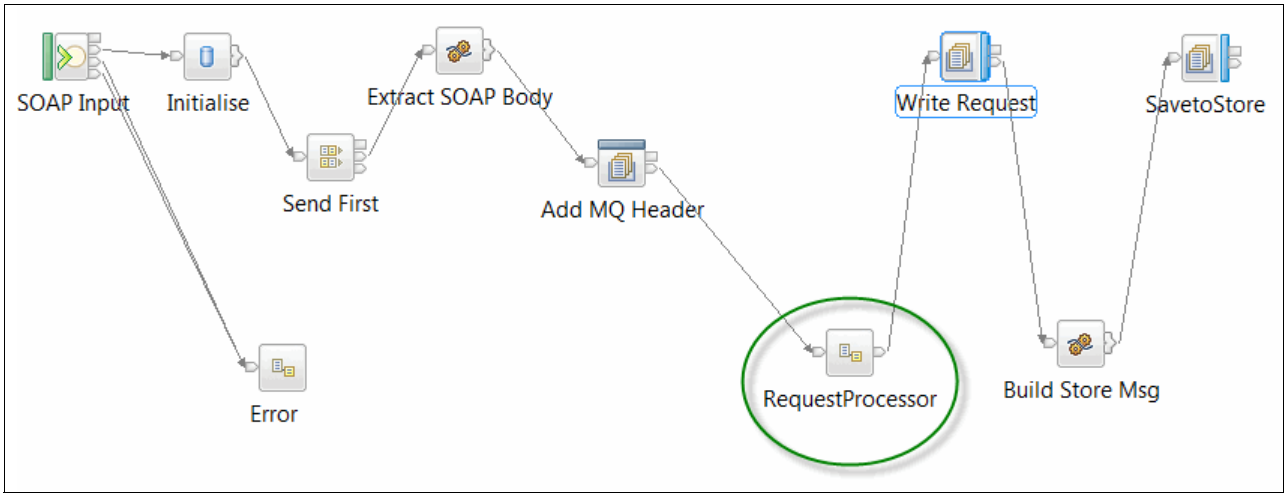

Figure 4-32 Generated request flow

In the Request Processor subflow, which is shown in Figure 4-33, a Java compute node is added that tries to get a response from the external cache. If a response does not exist in the cache, a request is sent to the external system. Otherwise, a response is sent immediately back to the client by using a SOAP Reply. Because the response message is taken as binary large object (BLOB) value from cache, a Reset Content Descriptor node is added, which associates the XMLNSC parser with the message.

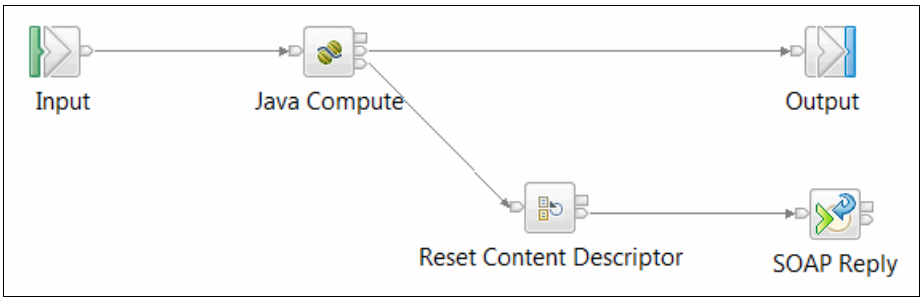

Figure 4-33 Generated subflow

Response flow

As in the request flow, the response flow includes a Response Processor subflow that allows the message to be modified, as shown in Figure 4-34. Other parts of the generated flow are left untouched.

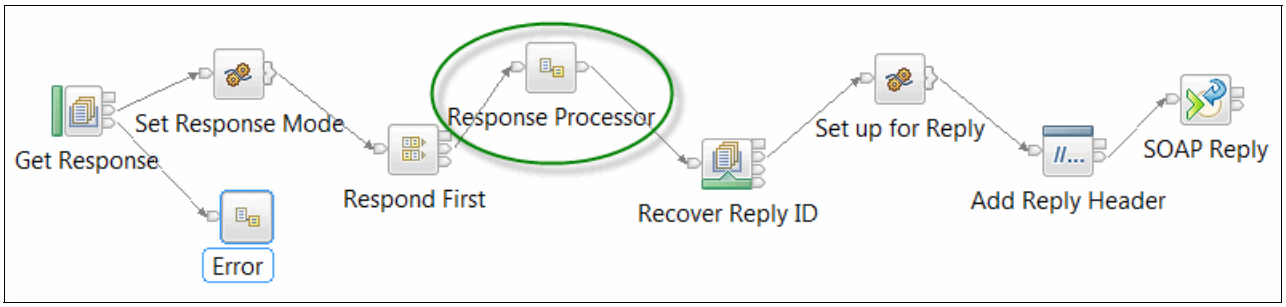

Figure 4-34 Generated response flow

In the Response Processor subflow, which is shown in Figure 4-35, there is one Java compute node. This node transparently inserts the value of the response into a cache.

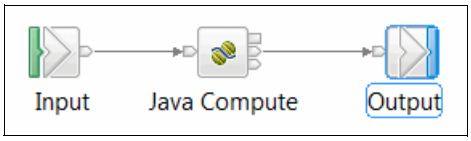

Figure 4-35 Generated subflow with added Java compute node

WebSphere eXtreme Scale client configuration

To use the WebSphere eXtreme Scale Java API, it must be made available to the code that is running in the Java compute nodes. Put the `ogclient.jar` file, which is available after the installation of the WebSphere eXtreme Scale client, in a *`Workpath`*`/shared-classes` folder of WebSphere Message Broker. The location of the folder depends on the operating system in which the broker runs:

- On Linux and UNIX systems: `/var/mqsi/common/profiles`
- On Windows XP and Windows 2003 Server systems: `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\common\profiles`

- On Windows 7 and Windows Server 2008 systems:
  `C:\ProgramData\IBM\MQSI\common\profiles`

Java code

The WebSphere Message Broker Toolkit automatically creates the Java classes that are used by the Java compute nodes. All of those classes are put in a separate project, which also can contain other related code.

Connecting to grid

A helper class named *GridProxy* is shown in Example 4-3. The GridProxy helper class is responsible for creating a connection to the WebSphere eXtreme Scale grid. A hardcoding of connect string is avoided by using ConfigurableService properties. More configuration can be done by using an XML file that describes the behavior of the near cache in the client. All Java nodes get access to the grid by using this helper class.

Inside the grid is a singleton property named *instance* that is initialized when the first call to a grid is made. Each time that access to the grid is needed, a `getSession` method is called, which returns a non-thread safe session object. This object must be closed (and returned to the poll) when all operation on the grid is completed.

A singleton pattern allows the grid to use a near cache feature in an efficient way, which is shared between code that uses put and get values from the grid.

Example 4-3 Proxy class that is used for connecting to a grid

```
public class GridProxy {

   private static GridProxy instance = new GridProxy();
   ObjectGrid grid = null;

    void initialize(String gridName){
      try {

      BrokerProxy b = BrokerProxy.getLocalInstance();
      while(!b.hasBeenPopulatedByBroker()) { Thread.sleep(100); }
      ConfigurableService myUDCS = b.getConfigurableService("UserDefined",
gridName);
      Properties props  = myUDCS.getProperties();

      ClientClusterContext ccc =
ObjectGridManagerFactory.getObjectGridManager().connect(props.getProperty("conn
ectString"), null,
            new URL("file:///"+props.getProperty("configFile")));
      grid = ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc,
gridName);

      System.out.println("Initalize GridProxy");
      } catch(Exception e) {
         e.printStackTrace();
      }
   }

    Session getSession(String gridName) throws TransactionCallbackException,
ObjectGridException{
       if(grid==null)
            initialize (gridName);
```

```
        returngrid.getSession();
    }
    static GridProxy getInstance() {

        return instance;
    }
}
```

You can set ConfigurableService properties from the command line by issuing the following commands:

```
mqsichangeproperties WMB8 -c UserDefined -o WMBGrid -n connectString -v
"172.17.218.198:2809"
mqsichangeproperties WMB8 -c UserDefined -o WMBGrid -n configFile -v
"C:/Dev/WXS/objectgrid.xml"
```

Alternatively, you can use WebSphere Message Broker Explorer to specify the properties, as shown in Figure 4-36.

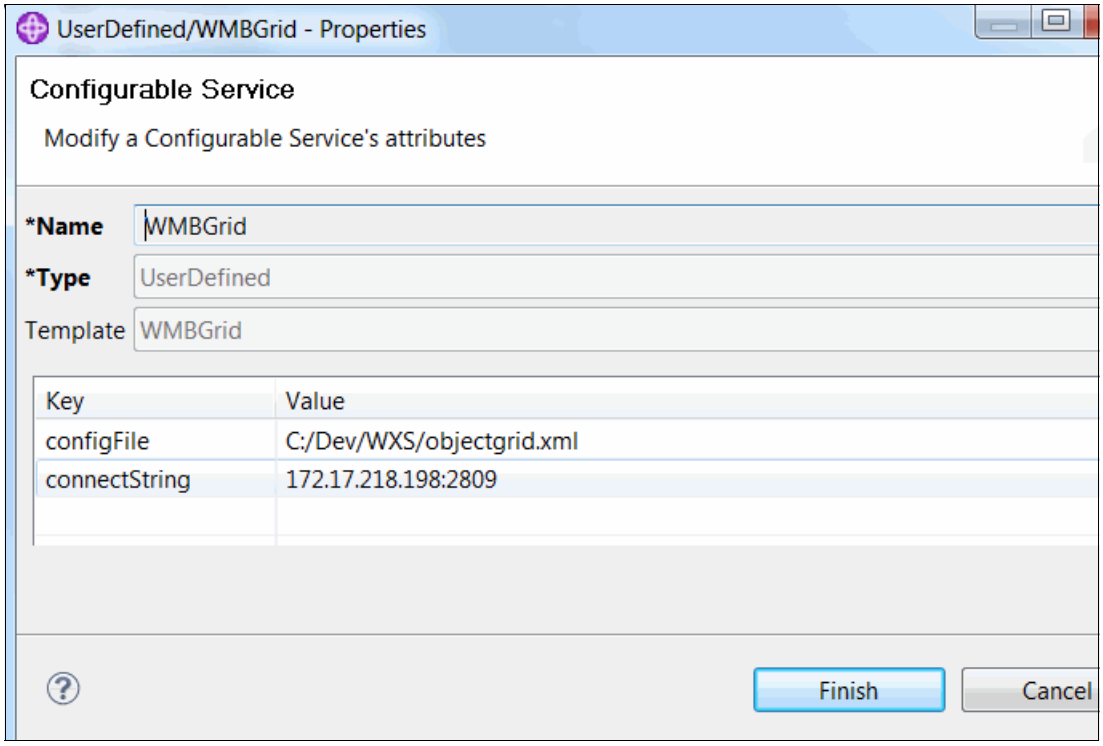

Figure 4-36 WebSphere Message Broker Explorer

Querying the cache

The cache is queried during the processing of a request. First, a session object is obtained from the helper class. (Session objects are not thread safe. Do not reuse them. The grid manager pools them automatically.) Then, the cache is queried by using a key from a request message. Choosing the correct fields for a key is important because incorrect keys can lead to false cache hits.

If no results are found, a `null` response is returned, and the message is sent to the back-end system. If data is found, a response message is created based on the result and forwarded to the `alt` terminal of the Java compute node. At the end, a session object is closed and returned to the pool.

Example 4-4 provides an example of a cache query.

Example 4-4 Example of querying a cache in Java compute node

```
try {
   Session sess = GridProxy.getInstance().getSession("WMBGrid");
   ObjectMap map = sess.getMap("WMBGrid");
   MbElement stId  =
inMessage.getRootElement().getLastChild().getLastChild().getFirstElementByPath("st
ationId");
   byte[] bytes  = (byte[]) map.get(stId.getValueAsString());
   if (bytes == null){
      out.propagate(inAssembly);
      }else {
      outMessage.getRootElement().createElementAsLastChildFromBitstream
                  (bytes,MbBLOB.PARSER_NAME,"","","",0,0,0);
      alt.propagate(outAssembly);
            }
   sess.close();
} catch (Exception e){
   e.printStackTrace();
   out.propagate(inAssembly);
}
```

Putting values into a cache

Writing to a cache is simple. After you obtain a session object, a cache key is read from a message body and used to put the entry into a cache. If an exception is thrown, such as in the case where the remote cache is not available, the message is propagated to the output terminal but the cache is not updated.

Example 4-5 provides an example of writing to the cache.

Example 4-5 Example of writing to a cache in Java compute node

```
try {
   Session sess = GridProxy.getInstance().getSession("WMBGrid");
   ObjectMap map = sess.getMap("WMBGrid");
   MbElement stId  =
message.getRootElement().getLastChild().getLastChild().getFirstElementByPath("stat
ionId");
   MbElement msgBody = message.getRootElement().getLastChild();
   byte[] body= msgBody.toBitstream("","","",0,0,0);
   map.put(stId.getValueAsString(),body);
   out.propagate(assembly);
} catch (Exception e){
   e.printStackTrace();
   out.propagate(assembly);
}
```

Stale values in cache

WebSphere Message Broker as an ESB has no control over the changes that happen in the back-end system. It is possible to have a response with cached values even if the data that is used as a base for a response changes.

Here are different strategies to deal with this type of scenario:

- Evictors

  Data in cache can be deleted after a period (which can be hours or even days). Eviction is based on time from creation, last access, or last update to an object.

- Detecting changes in the back end

  Data is usually changed in a database, and by using, for example, Database Input node in WebSphere Message Broker. You can (for each changed row) delete it from a cache and force the storing of a new response when a request arrives.

Although each strategy has its benefits and should be applied based on the scenario, using a change detection shortens the window during which stale responses are returned to a client.

Near cache

WebSphere eXtreme Scale has a built-in near-cache functionality. Thus, responses from cache are storied internally in the Java virtual machine (JVM) where the client runs, without the need to go to the external cache. Near-cache is created on every connection and is destroyed when the connection is closed. Because this cache is only temporary and linked with the connected process, it is not possible to delete staled cache entries that are based on event detection. Also, there are several risks that are associated with it:

- Out-of-memory problem if there is no eviction set and the process runs for several days
- Random stale reads

To avoid these problems, consider setting a custom eviction time or disable near-cache.

During the setup of connection to a WebSphere eXtreme Scale cache, it is possible to override the object grid configuration file, `objectgrid.xml`. To disable the near-cache, set the `numberOfBuckets` parameter for the map that is used to `0`. If this parameter has a non-zero value, the evictor must be set with a reasonably short `timeToLive` value, as shown in Example 4-6.

Example 4-6 Example of disabling near-cache

```
<objectGridConfig xsi:schemaLocation="http://ibm.com/ws/objectgrid/config
../objectGrid.xsd">
   <objectGrids>
      <objectGrid name="WMBGrid" txTimeout="30"><customProperty
name="capacityLimit" value="-1"/>
         <backingMap name="WMBGrid" readOnly="false" lockStrategy="NONE"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="10" copyMode="COPY_TO_BYTES"
numberOfBuckets="0"/>
      </objectGrid>
   </objectGrids>
</objectGridConfig>
```

For information about the parameters in the `objectgrid.xml` file, see 6.2.3, “Understanding WebSphere eXtreme Scale parameters” on page 108.

4.3.5 Conclusion

In the test environment used in writing this book, the initial solution was able to achieve up to 50 transaction per second with a 300 ms response time, as a benchmark. After implementing the external cache solution in WebSphere, the following benefits were seen:

- Significantly decreased response time, which leads to better client satisfaction when you use social media and interacting with service. In our test environment, the response time decreased from 300 ms to 10 ms.
- Significant increase in transactions per second, which means that there are no timeouts when a service is accessed by an external system. In our test environment, our transactions per second increased from 50 to 500.
- Decreased load on a target ticketing system, which means that there are no delays during ticket sales during peak times. In our test environment, the load decreased from 90% to 50%.

These numbers indicate what was achieved in a small test environment and are provided as an example compared to our initial benchmark.

5

Enterprise caching in an extreme transaction processing environment

There are many benefits of using caching technology to achieve extreme transaction processing (XTP). XTP is achieved by providing fast data access and the ability to process high volumes of data in short periods of time. A stock trading application environment is used as a use case. However, the same principles of design and implementation can be applied to other environments that require high-volume data processing.

The following topics are described:

5.1 Customer problem

A fictional stock trading company buys and sells stocks on behalf of its customers. The company faces a problem because of the high volume of orders that are placed by customers each day, and the short time window to run batches of orders before the prices of the ordered stocks change. New prices can arrive in intervals of milliseconds, so the speed of order execution is critical to the accuracy of transactions and client satisfaction.

In this example scenario, the company is looking to replace its existing order processing unit with a solution that allows large volumes of orders to be ran in the shortest amount of time. In addition, the proposed solution must be scalable to accommodate future growth in order volume.

5.2 Solution

To solve the client problem, a Java application that uses WebSphere eXtreme Scale as the main data repository is proposed. Caching technology, which allows low data-access times, is used to compute high volumes of data quickly. And, WebSphere eXtreme Scale agent technology is used to keep computation local to the data.

5.2.1 Application environment

The stock trading application consolidates data that is received from two sources.

The first source is a stream of client orders to buy or sell stocks, which are typically submitted through a public interface such as a website. Order volume depends on the size of the company's client base and the changing status of the stock market. Other factors that affect order volume include the time of day and even the day of the week or month.

The second source of data is the stream of TICs that provide the latest individual stock prices and are streamed to the company from the various stock exchanges through a messaging interface.

TIC: A *TIC* is a piece of atomic information about the current price of an individual stock, which is based on the latest transactions of that stock. Its name comes from the sound that is made by mechanical telegraph machines that were used to send gold prices in the late 19th and early 20th centuries.

The volume of messages in both streams might be high. The fluctuation in message volume might be frequent. When stock prices change rapidly, more TICs are sent from the stock exchange and often, more orders are filed by customers.

Figure 5-1 presents a simplified topology of a stock trading application. It does not contain all of the processing units that are found in actual applications and is simplified to provide an understanding of the example scenario that is presented here.

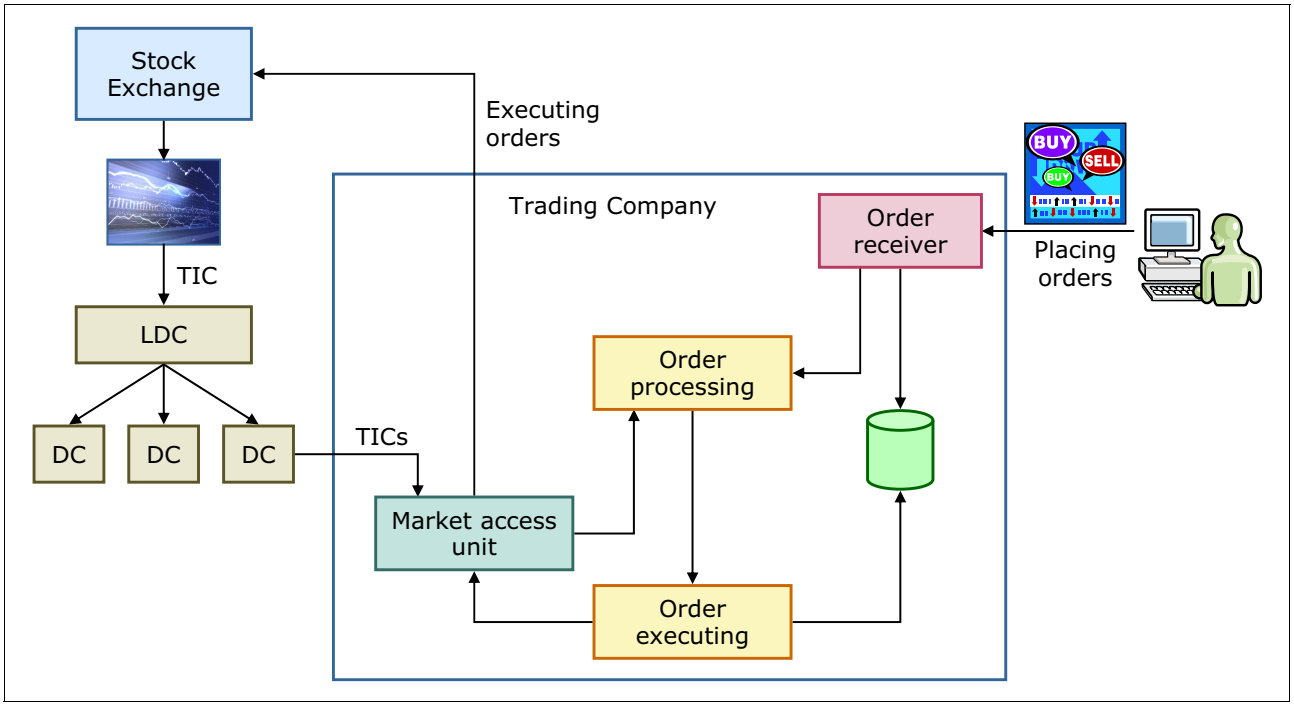

Figure 5-1 Stock trading application, high-level view

As shown in Figure 5-1, client orders arrive from a web interface to an order receiver unit. The order receiver unit stores the order records in a database and sends them to an order processing unit. At the same time, the stock exchange streams the TICs to a local data center (LDC). This center is located a short distance (less than 10 miles) from the stock exchange location. Newly arrived TICs (represented with *TIC* in the figure) are replicated to remote data centers (DCs) that are local to the stock trading applications. The stock trading applications receive the TICs through the market access unit, which forwards the TICs to the order processing unit. Based on the most recent TIC for each stock, the order processing unit sets the strike price on all orders waiting for processing. All processed orders are sent back to the stock exchange through the market access unit by the order execution unit. At the same time, the order records in the database are updated with the strike price.

5.2.2 Application requirements

The application must have the capacity to process high volumes of incoming orders and high volumes of incoming TICs. In addition, it has to be scalable for times when the volume of orders and TICs might increase. Such times include so-called boundary situations, which are the periods of time around major market corrections or global events. There might also be increased volumes of orders before holiday periods, as many short-term stock holders offload their holdings before they go on vacation. The Stock Trading Company must be able to adjust the computing capacity of the application for each trading day. The day starts at the beginning of the trading session and ends shortly after the session closes.

5.2.3 Proposed technology

The proposed solution uses WebSphere eXtreme Scale caching technology to provide a highly scalable, XTP-level solution for the stock trading application. Two caching technologies that are used in the proposed solution are the WebSphere eXtreme Scale cache grid and WebSphere eXtreme Scale agents.

WebSphere eXtreme Scale cache grid

WebSphere eXtreme Scale cache grid is a distributed data cache that provides a high level of data availability, failover protection, unlimited scalability immediately, and low access times. The grid is composed of distributed data containers that contain fragments of data sets along with replicas of the data.

The primary data values and their replicas are remote to each other and are distributed in such a way that, in the event of a server failure, the data is reallocated among the remaining containers. Also, replica data is used to recover any data that is lost with the failing server. In the same way, adding a container increases total grid capacity and causes the data to be reallocated for even data distribution among all of the containers. Because the cache uses memory, fast access times that far exceed the times of traditional database systems are assured.

WebSphere eXtreme Scale agents

The WebSphere eXtreme Scale agent technology provides public Java interfaces to implement agents that can manipulate data that is in the WebSphere eXtreme Scale cache within the cache itself. In other words, the computation of the data is done in-memory on the server side of the cache infrastructure, speeding up data access and processing.

Each implementation of an agent is associated with a specific map, and when the agent implementation is run, it can easily access the data in the map. Even if the map is distributed across multiple partitions, the agent computations for a specific map occur at each partition, as the agent is routed to every partition where the map is located. This process is transparent to the user of agent application programming interfaces (APIs).

There are two public Java Interfaces that use the WebSphere eXtreme Scale agent technology:

- `com.ibm.websphere.objectgrid.datagrid.MapGridAgent`
- `com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent`

The MapGridAgent class defines APIs that provide the means to query and potentially manipulate the content of the map with which the agent is associated. One of the APIs, shown in Example 5-1, is used to retrieve new orders from the Order Repository cache.

Example 5-1 processAllEntries() API of the MapGridAgent

```
/**
* This method is called once per partition.
* Filter the set of entries by setting values for: sourceContainer, partition,
* and targetContainer
*
* @param s Session to use for work. Transactions are controlled by the caller.
* @param map ObjectMap this agent was invoked for.
* @return A map of processed values for keys
*/
```

Example 5-2 shows an example API of the MapGridAgent.

Example 5-2 processAllEntries() API of the MapGridAgent

```
public Map<String, OrderRecord> processAllEntries(Session s, ObjectMap map)
```

The ReduceGridAgent class defines APIs that provide the means to access the content of the map and return a product of actions on this content in a form of a single Java object. One of the APIs, shown in Example 5-3, is used to update orders that are waiting for processing in the Order Queue cache.

Example 5-3 reduce() API of the ReduceGridAgent

```
/**
* This method is called on each partition. The Agent may have extra state
* including possible a query to eliminate some partition entries.
*
* @param s Session to use for work. Transactions are controlled by the caller.
* @param map The ObjectMap this agent was invoked for
* @return The value calculated from processing the entries for the partition.
*/
public Object reduce(Session s, ObjectMap map)
```

The proposed solution uses both types of WebSphere eXtreme Scale agents for their unique capabilities.

5.2.4 Implementation

The proposed solution replaces the order processing unit, which is shown in Figure 5-1 on page 85, with a Java application that uses WebSphere eXtreme Scale technology. The integration between the new order processing application and other components of the stock trading application is accomplished through a shared eXtreme Scale cache and through IBM WebSphere MQ infrastructure. Source code for this scenario is provided in “Additional files for Chapter 3” on page 135.

The system design of the new application is shown in Figure 5-2.

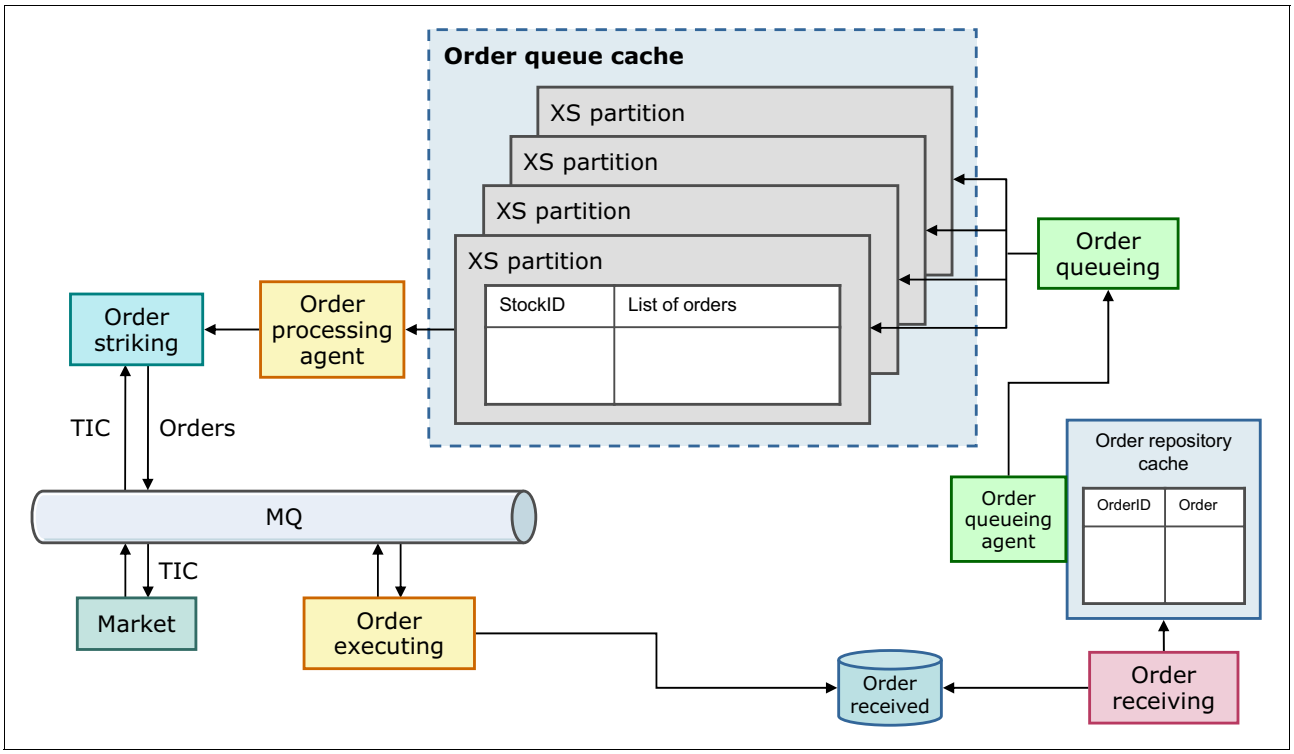

Figure 5-2 Stock order processing with a new application by using eXtreme Scale technology

There are two cache grids that contain the data that is used by the application: the order repository cache (`GridOR`) and order queue cache (`GridOQ`). They are defined in a descriptor file, which is shown in Example 5-4. The copy mode defines what data value instance the client application receives when getting an entry from the map. For maximum performance, the copy mode of the GridOR is set to NO_COPY. The data in the grid is not modified and the grid serves only as a buffer for new orders. For information about the ObjectGrid descriptor file parameters in the `objectgrid.xml` file, see 6.2.3, “Understanding WebSphere eXtreme Scale parameters” on page 108.

Example 5-4 ObjectGrid descriptor file

```
<objectGrids>
        <objectGrid name="GridOR">
      <backingMap name="OrdersRepository" lockStrategy="PESSIMISTIC"
copyMode="NO_COPY"/>
   </objectGrid>
   <objectGrid name="GridOQ">
      <backingMap name="OrdersQueueAF" lockStrategy="PESSIMISTIC"/>
      <backingMap name="OrdersQueueGL" lockStrategy="PESSIMISTIC"/>
      <backingMap name="OrdersQueueMS" lockStrategy="PESSIMISTIC"/>
      <backingMap name="OrdersQueueTZ" lockStrategy="PESSIMISTIC"/>
   </objectGrid>
</objectGrids>
```

The order repository cache that is implemented as `GridOR` is a side cache that is populated by the existing order receiving unit. New orders are placed in the cache as they arrive. The orders are defined by the Java class OrderRecord, with fields as listed in Example 5-5.

Example 5-5 OrderRecord field definitions

```
   public static int RECEIVED = 1;
   public static int QUEUED = 2;
   public static int PROCESSED = 3;
   //Value fields
   private String orderID;
   private long customerID;
   private String stockID;
   private long dateOfOrder;
   private int numberOfStocks;
   private boolean buy = true;
   private int strikingPrice = -1;
```

The order queue cache that is implemented as `GridOQ` contains four maps (for some tests eight maps were used) that contain order queues for all stocks. During testing of the solution only 24 stock IDs were defined by using letters of the Latin alphabet. The stock IDs are evenly distributed across the four maps, with six names in each map. To achieve high performance when you update orders in the order queue cache, the maps in this cache are deployed in such a way that each map is contained in one partition. The corresponding deployment file is shown in Example 5-6.

Example 5-6 Deployment policy descriptor file

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

   <objectgridDeployment objectgridName="GridOR">
         <mapSet name="mapSetOR" numberOfPartitions="13" minSyncReplicas="0"
maxSyncReplicas="1" >
            <map ref="OrdersRepository"/>
         </mapSet>
   </objectgridDeployment>
      <objectgridDeployment objectgridName="GridOQ">
         <mapSet name="mapSetOQ" numberOfPartitions="1" minSyncReplicas="0"
maxSyncReplicas="1" >
            <map ref="OrdersQueueAF"/>
            <map ref="OrdersQueueGL"/>
            <map ref="OrdersQueueMS"/>
            <map ref="OrdersQueueTZ"/>
         </mapSet>
   </objectgridDeployment>

</deploymentPolicy>
```

The order queuing unit uses the OrderQueueingAgent, which is an implementation of the `com.ibm.websphere.objectgrid.datagrid.MapGridAgent`, to extract the new orders from the order repository cache and to clean the cache. See Example 5-7.

Example 5-7 OrderQueueingAgent code

```
public Map<String, OrderRecord> processAllEntries(Session s, ObjectMap map){
   String methodName = "processAllEntries";
   HashMap<String, OrderRecord> l = new HashMap<String, OrderRecord>();
   try {
      MapIndex allIndex = (MapIndex)
map.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
      Iterator keys = allIndex.findAll();
      while (keys.hasNext()) {
         Object key0 = keys.next();
         String key = (key0 instanceof String) ? (String) key0 : null;
         if ( key != null) {
            Object value0 = map.get(key);
            OrderRecord value =  (value0 instanceof OrderRecord) ? (OrderRecord)
value0 : null;
            if(value != null && value.getState() == OrderRecord.RECEIVED){
               if(filterOn){
                  if(stockID != null && !stockID.equals(value.getStockID())){
                     continue;
                  }
               }
               l.put(key, value);
               map.remove(key);
            }
         }
      }
   } catch (ObjectGridException e) {
      FFDCFilter.processException(e, CLASS_NAME + ".processAllEntries","137");
      if (tc.isDebugEnabled()) { e.printStackTrace(System.out); }
   }
   return l;
}
```

After the new orders are extracted and the cache is cleaned, the order queuing unit organizes new orders into queues that are based on the stock ID and puts those queues in the order queue cache. The core functionality of the order queuing unit is shown in Example 5-8.

Example 5-8 Order Queueing code

```
Map m = map.getAgentManager().callMapAgent(new OrderQueueingAgent());
int setSize = m.keySet().size();
if(setSize > 0){
   for (Object key: m.keySet()) {
      if ( key != null) {
         Object value0 = m.get(key);
         OrderRecord  value = (value0 instanceof OrderRecord) ? (OrderRecord)
value0 : null;
         if(value != null){
            value.setState(OrderRecord.QUEUED);
            addOrder2LocalQueue(value);
         }
```

```
      }
   }
   uploadQueues(sessQ);
}

where:

public static void addOrder2LocalQueue(OrderRecord value){
   Object queueO = queues.get(value.getStockID());
   if(queueO != null){
      ArrayList<OrderRecord> queue = queueO instanceof ArrayList ?
(ArrayList)queueO : null;
      if(queue != null){
         queue.add(value);
      }
   }
}

and:

public static void uploadQueues(Session sess) throws Exception {
   Iterator<String> iter = queues.keySet().iterator();
   while(iter.hasNext()){
      String queueName = (String)iter.next();
      String queueMapName = MyCompConstants.getQueueMapName(queueName);
      ObjectMap map = sess.getMap(queueMapName);
      Object qO = map.get(queueName);
      if(qO == null){
         map.insert(queueName, (ArrayList)queues.get(queueName));
      } else {
         ((ArrayList)qO).addAll((ArrayList)queues.get(queueName));
         map.remove(queueName);
         map.insert(queueName, (ArrayList)qO);
      }
      queues.put(queueName, new ArrayList());
   }
}
```

The order processing unit gets new incoming TICs from the message queue and, based on those TICs, updates the orders for corresponding stocks in the order queue cache by using the OrderProcessingAgent, which implements the `com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent`, as shown in Example 5-9. The agent updates all orders for a specified stock ID with the strike price and a new state.

Example 5-9 OrderProcessingAgent code

```
public Object reduce(Session s, ObjectMap map){
   String methodName = "reduce";
   int numberOrdersProcessed = 0;
   try {
      MapIndex allIndex = (MapIndex)
map.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
      Iterator keys = allIndex.findAll();
      while (keys.hasNext()) {
         Object keyO = keys.next();
         String key = (keyO instanceof String) ? (String) keyO : null;
```

```
         if ( key != null && key.equals(stockID)) {
            Object value0 = map.get(key);
            List list =  (value0 instanceof List) ? (List) value0 : null;
            if(list != null && list.size() > 0){
               for (int i=0; i<list.size();i++){
                  Object order0 = list.get(i);
                  if(order0 != null){
                     OrderRecord order = order0 instanceof OrderRecord ?
(OrderRecord)order0 : null;
                     if(order != null){
                        order.setStrikingPrice(strikingPrice);
                        order.setState(OrderRecord.PROCESSED);
                     }
                  }
               }
               map.update(key0, list);
            }
         }
      }
   } catch (ObjectGridException e) {
      e.printStackTrace();
   }
   return numberOrdersProcessed;
}
```

The order processing unit calls the OrderProcessingRunnable agent for each new TIC that is received from WebSphere MQ. Example 5-10 shows the work that is done by the order processing unit.

Example 5-10 Order Processing unit code

```
long start = System.currentTimeMillis();
long end = start;
DateFormat df = new SimpleDateFormat("HH:mm:ss");
Date time = new Date(start);
while (df.format(time).compareTo(sessionClosingTime.toString())<0){
   try {
      queue.get(receiveMessage, getMessageOptions);
      Object TIC0 = receiveMessage.readObject();
      if(TIC0 != null){
         TIC tic = TIC0 instanceof TIC ? (TIC)TIC0 : null;
         if(tic != null){
            String stockID = tic.getStockID();
            latestPrices.put(stockID, tic.getStockPrice());
            orderProcessingRunnable = new OrderProcessingRunnable(tic, grid);
            try {
               threadPool.execute(orderProcessingRunnable);
            } catch ( Exception e ) {
               e.printStackTrace();
            }
         }
         end = System.currentTimeMillis();
         receiveMessage = new MQMessage();
      }
   } catch (MQException e){
      e.printStackTrace();
```

```
   }
   time = new Date(end);
}

closeSession(latestPrices,grid);
```

The OrderProcessingRunnable agent calls the setStrikingPrice() API shown in Example 5-11.

Example 5-11 setStrikingPrice() API

```
public static int setStrikingPrice(String stockID, int price, ObjectGrid grid)
throws Exception {
   Session sess = grid.getSession();
   ObjectMap map = sess.getMap(MyCompConstants.getQueueMapName(stockID));
   OrderProcessingAgent agent = new OrderProcessingAgent();
   agent.setStockID(stockID);
   agent.setStrikingPrice(price);
   int numberOrders = (Integer)map.getAgentManager().callReduceAgent(agent);
   Object orderQueueO = map.get(stockID);
   map.remove(stockID);
   if(orderQueueO != null){
      List<OrderRecord> orderQueue = orderQueueO instanceof List ?
(List)orderQueueO : null;
      if(orderQueue != null){
         sendOffRecords((OrderRecord[])orderQueue.toArray());
      }
   }
   sess.close();
   return numberOrders;
}
```

The API creates an instance of the OrderProcessingAgent and starts the agent for a specified stock ID and strike price. The API then calls the sendOffRecords() API, shown in Example 5-12. The sendOffRecords() API sends the orders into a dedicated WebSphere MQ message queue.

Example 5-12 sendOffRecords() API

```
public static void sendOffRecords(OrderRecord[] records){
   MQQueueManager queueManager = null;
   try {
      MQEnvironment.hostname = MyCompConstants.MQ_MANAGER_HOST;
      MQEnvironment.channel = MyCompConstants.ORDERS_MQ_CHANNEL;
      queueManager=new MQQueueManager(MyCompConstants.MQ_MANAGER);
      int openOptions = MQConstants.MQOO_INPUT_AS_Q_DEF | MQConstants.MQOO_OUTPUT;
      MQQueue queue = queueManager.accessQueue(MyCompConstants.ORDERS_QUEUE_NAME,
openOptions);

      MQPutMessageOptions pmo = new MQPutMessageOptions();
      for(int i=0; i< records.length; i++){
         MQMessage msg = new MQMessage();
         msg.writeObject(records[i]);
         queue.put(msg, pmo);
         queue.close();
      }
      queueManager.disconnect();
```

```
   } catch (Exception e){
      e.printStackTrace();
   }
}
```

The order executing module that is a part of the existing application picks up the processed orders from the WebSphere MQ queue. The module runs them by sending orders to the Stock Exchange and by updating order records in the database.

5.2.5 Simulating the client scenario

This section describes how the solution is verified by showing the performance gains that help resolve the client problem. For performance validation, the production environment is simulated in a test lab.

Environment

To simulate the incoming stream of client orders, the order receiving application is implemented. The application populates the order registry cache with a sequence of incoming orders that are randomly generated in batches. The volume of the new-order stream can be manipulated. The incoming stream of TICs is simulated by the TIC receiving application, which simulates a daily stock trading session. Initial prices for all stocks are generated at the beginning of the session, and later, random changes to all stocks prices are generated. The volume of the TIC flow can be manipulated, too.

Figure 5-3 shows the hardware environment that is used in this simulated environment. Within a single physical server, five identical Linux virtual machines (VMs) were deployed, each running Linux Red Hat Version 6. For the book, each server used a single processor and 4 GB of RAM.

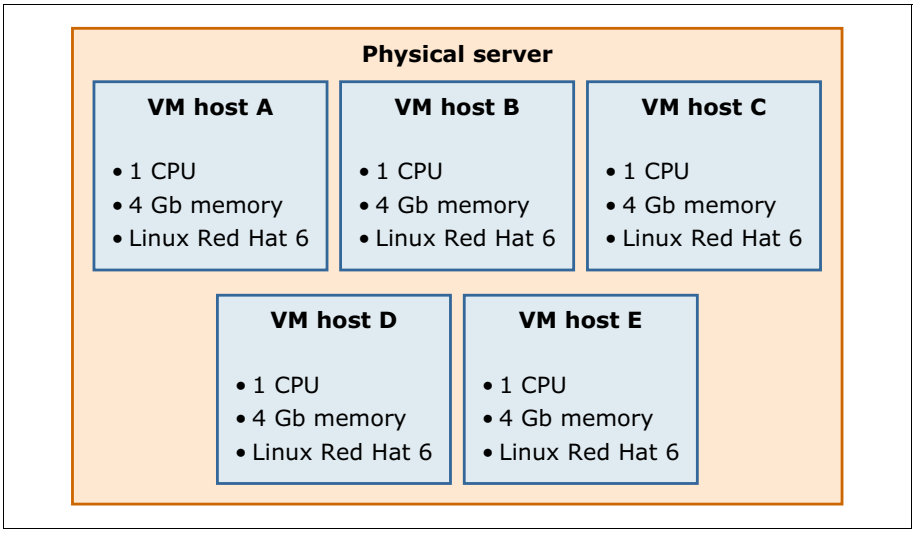

Figure 5-3 Hardware topology of a client solution

The end-to-end scenario was run three times by using the same physical hardware. The initial run served as a baseline. Then, different topologies and configurations were used to find the configuration that gave the best results in terms of processing capacity per unit of time. In addition, some resource settings for VMs were changed to get the best performance for the application.

Baseline

For the baseline run, the WebSphere eXtreme Scale catalog and WebSphere eXtreme Scale console ran on Host A. And, four eXtreme Scale containers ran on two other hosts, Host C and Host D. Host C hosts receiver orders and queue orders and Host B hosts process orders and TIC receiving. The configuration is shown in Figure 5-4.

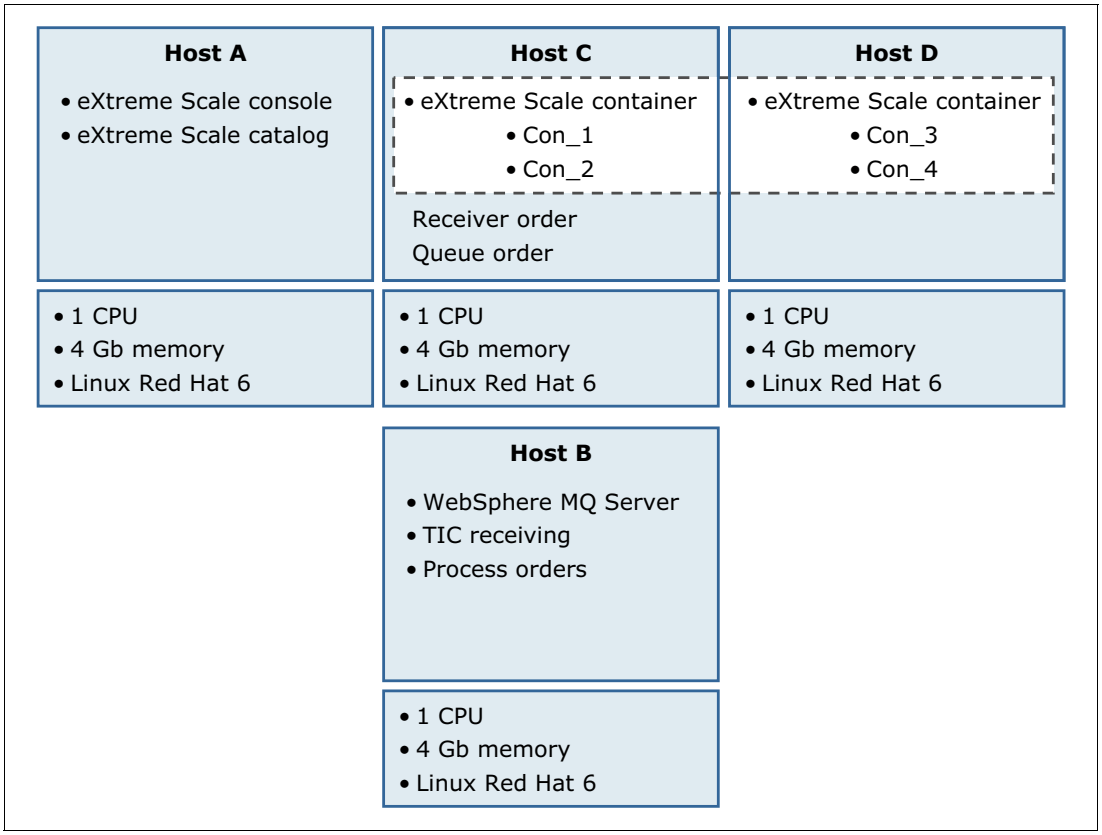

Figure 5-4 Environment for baseline

During the baseline run, 846 orders per second were loaded into the order records grid and queued into the order queues grid, as illustrated in Figure 5-5. This baseline is used for comparison purposes only throughout the rest of this scenario. This data is purely for demonstration purposes to show the performance gains that are seen in our test environment.

Current summary over last 30 seconds

Used Capacity	Number of cache entries	Average Throughput	Average Transaction Time
50.83 KB	105	846.07 transactions/second	0.83 milliseconds

Figure 5-5 Baseline transactions details

The process orders module was able to process all incoming orders and insert them into the message queue. This process made them available for order execution and to be committed at the stock exchange.

Next, a small change was made in the topology: the WebSphere eXtreme Scale containers that were running on Hosts C and D were stopped and started on Host A. This change placed all components on a single server, Host A. However, the WebSphere MQ server, process orders, and TIC receiving were still running on the other host, Host B. The resulting configuration is shown in Figure 5-6.

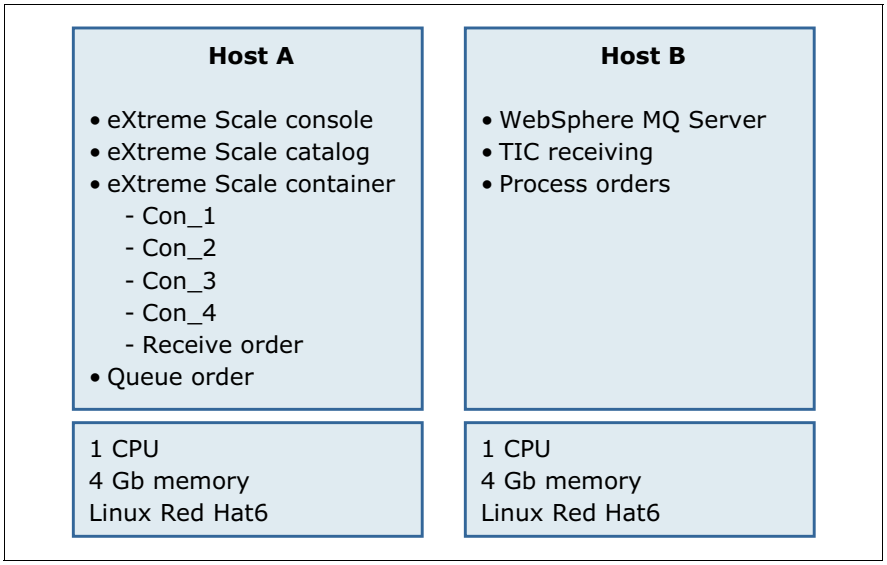

Figure 5-6 Server details

For this configuration, the flow of incoming orders was the same as before the change, where the cache was distributed between multiple servers. Placement of all components on one host caused high levels of CPU usage for this host. Figure 5-7 shows the high usage of CPU on Host A within our test environment. In our test environment, more than 80% of the CPU and 77% of memory is used by Java applications. The entire volume of incoming orders was successfully processed and sent to WebSphere MQ.

```
top - 10:19:52 up 5 days,  3:01, 11 users,  load average: 3.64, 4.12, 3.78
Tasks: 189 total,   1 running, 188 sleeping,   0 stopped,   0 zombie
Cpu(s): 80.5%us, 17.5%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  1.9%si,  0.0%st
Mem:   3808664k total,  2941376k used,   867288k free,   166268k buffers
Swap:  4128760k total,        0k used,  4128760k free,  1226776k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
19482 root      20   0  732m 184m 9464 S 23.8  5.0   6:36.14 java
20187 root      20   0  731m 150m 9488 S 19.2  4.1   4:41.54 java
20707 root      20   0  735m 147m 9488 S 16.6  4.0   3:52.41 java
19850 root      20   0  719m 130m 9476 S 16.3  3.5   3:38.27 java
24009 root      20   0  634m  64m 9420 S 10.4  1.7   1:11.51 java
24310 root      20   0  681m 105m 9412 S 10.4  2.8   1:10.21 java
22795 root      20   0  732m 151m 9552 S  0.7  4.1   2:04.15 java
   20 root      20   0     0    0    0 S  0.3  0.0   1:33.10 ata/0
 5257 itsouser  20   0  400m   9m 6148 S  0.3  0.3   7:11.91 vmtoolsd
```

Figure 5-7 High CPU usage by Java applications

Resource adjustment

To accommodate for the higher resource requirements, the hardware specifications were being changed to better accommodate the environment needs. The number of processors of the WebSphere eXtreme Scale server (Host A) was increased to eight and the memory was

increased to 20 GB. On the WebSphere MQ server (Host B), the number of processors was increased to four.

During this test, the entire eXtremeScale instance was running on a single server, along with the receive orders and the queue orders applications. On the other server, the WebSphere MQ server, TIC receiving, and process orders were running.

This test used the architecture that is presented in Figure 5-8.

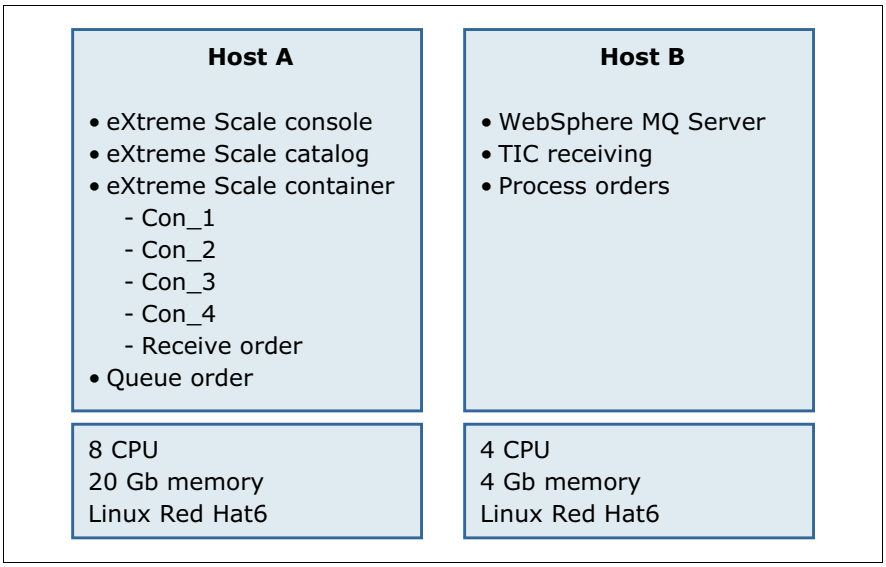

Figure 5-8 Test two: hardware and application architecture

During this test, it was possible to verify that the system resources were sufficient, and with that verification, the rate of incoming new orders increased. The rate reached 1,460 transactions per second, as shown in Figure 5-9. This amount is an increase in throughput of 66% in our test environment when compared to our baseline.

Current summary over last 30 seconds

Used Capacity	Number of cache entries	Average Throughput	Average Transaction Time
53.23 KB	110	1460.92 transactions/second	0.45 milliseconds

Figure 5-9 Test two: Average throughput

The CPU usage was monitored throughout the test and remained stable and lower than 40%, as shown in Figure 5-10.

```
top - 04:12:04 up 1 day, 17:49, 16 users,  load average: 0.93, 0.67, 0.27
Tasks: 285 total,   1 running, 284 sleeping,   0 stopped,   0 zombie
Cpu(s): 23.9%us,  4.1%sy,  0.0%ni, 71.4%id,  0.0%wa,  0.0%hi,  0.6%si,  0.0%st
Mem:  20470092k total,  2532460k used, 17937632k free,   205788k buffers
Swap:  4128760k total,        0k used,  4128760k free,   408184k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3652 root      20   0  792m 242m 9496 S 84.7  1.2   2:39.89 java
 4653 root      20   0  839m 259m 9504 S 50.2  1.3   1:57.96 java
 6671 root      20   0  704m 108m 9452 S 43.5  0.5   1:22.95 java
 3992 root      20   0  783m 179m 9496 S 25.2  0.9   1:10.67 java
 4334 root      20   0  755m 151m 9496 S 25.2  0.8   1:09.13 java
 5664 root      20   0  686m 115m 9448 S 22.9  0.6   0:50.09 java
 2310 root      20   0 15160 1376  936 R  0.3  0.0   0:01.55 top
    1 root      20   0 19356 1572 1248 S  0.0  0.0   0:01.86 init
```

Figure 5-10 Test Two: CPU usage

The Java applications were still putting a high load on CPU. However, as the hardware resources were available to do the processing, it allowed better order processing performance than in the first test. All new incoming orders were being processed and sent to the queue. The hardware upgrade allowed more transactions per second.

Maximizing performance

To demonstrate the flexibility and scalability of the application in terms of processing capability, the rate of new incoming orders was multiplied by 10 times when compared to the baseline. To increase the incoming orders volume, multiple receive orders applications were used to simulate an overload of client requests.

Five servers are used:

- Host A had eight CPUs and 20 GB of memory and served as the host for the WebSphere eXtreme Scale console, WebSphere eXtreme Scale catalog, four WebSphere eXtreme Scale containers, a receive order application, and a queue order application.
- Host B had four CPUs and 4 GB of memory and ran IBM WebSphere MQ, TIC receiving, and process orders.
- Host C, Host D, and Host E were configured to be one CPU with 4 GB of memory, with three instances of receive orders running on each server.

The server architecture is shown in Figure 5-11.

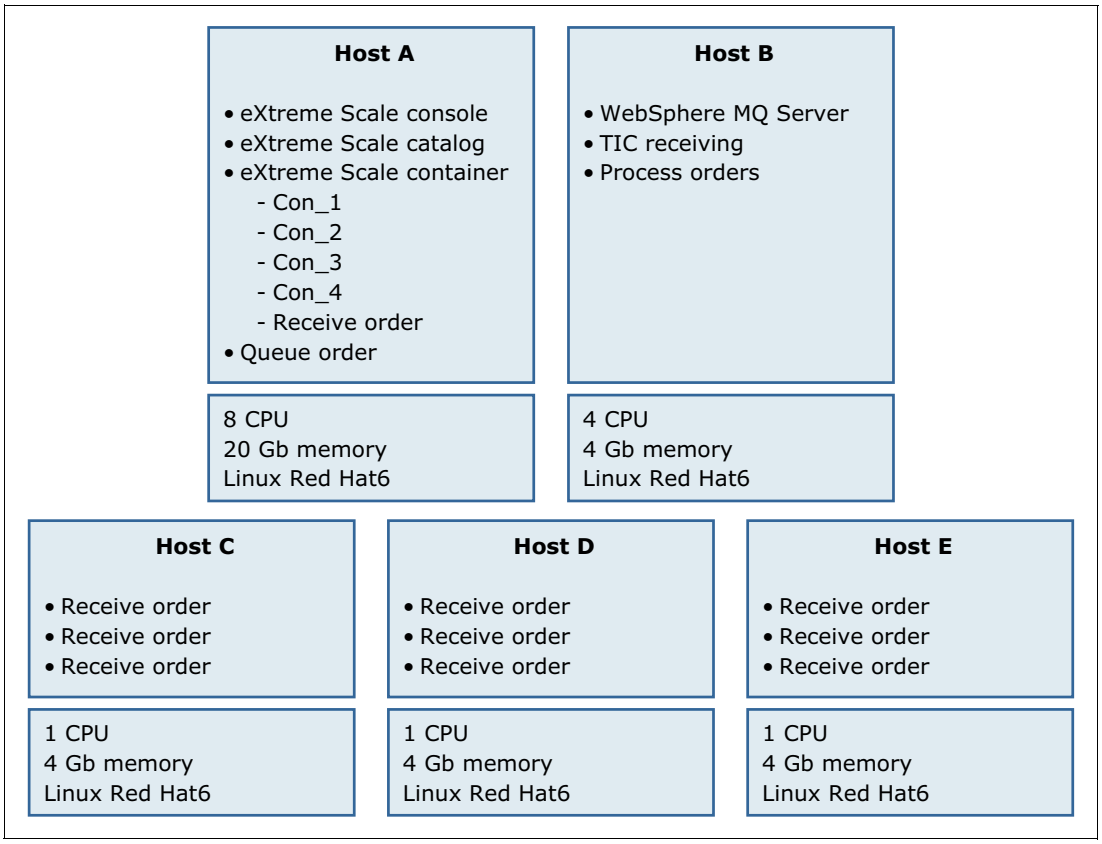

Figure 5-11 Test three: Hardware and application architecture

WebSphere MQ server was initialized, and then WebSphere eXtreme Scale was started, followed by the applications. The load was increased gradually by adding new receive order applications. With this increase, it was possible to get the average throughput values for increasing load values.

After starting all 10 receive orders applications, it was possible to reach an average throughput of 5,125 transactions per second in our test environment, as shown in Figure 5-12.

Current summary over last 30 seconds

Used Capacity	Number of cache entries	Average Throughput	Average Transaction Time
86.23 KB	180	5125.30 transactions/second	0.59 milliseconds

Figure 5-12 Test three: Average throughput

During this run, the CPU usage in our test environment was constantly above 60% and memory usage was lower than 70%, as shown in Figure 5-13.

This status verified that the queue orders application can accept and work with the workload increase, putting more orders to the queues to be processed. Additionally, and more importantly, it was possible to verify that the processing orders application can process all incoming orders. This outcome proves that high processing performance can be achieved by the proposed application. See Figure 5-13.

```
top - 09:40:05 up 1 day, 23:17,  9 users,  load average: 1.90, 1.58, 1.29
Tasks: 271 total,   1 running, 270 sleeping,   0 stopped,   0 zombie
Cpu(s): 54.7%us, 13.7%sy,  0.0%ni, 29.2%id,  0.0%wa,  0.3%hi,  2.1%si,  0.0%st
Mem:  20470092k total, 11036908k used,  9433184k free,   221104k buffers
Swap:  4128760k total,        0k used,  4128760k free,  3538476k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
18656 root      20   0 2432m 2.0g 9516 S 186.0 10.5  39:29.54 java
19716 root      20   0 2397m 1.0g 9516 S 136.5  5.2  29:03.70 java
19068 root      20   0 2387m 514m 9508 S 115.9  2.6  23:23.09 java
19387 root      20   0 2434m 658m 9516 S 111.9  3.3  23:23.04 java
22068 root      20   0 2438m 1.5g 9348 S 31.2  7.9   7:42.12 java
20183 root      20   0 2416m 301m 9452 S 16.3  1.5   5:34.24 java
 3159 root      20   0  859m 268m 9608 S  0.7  1.3  22:53.19 java
 7897 root      20   0 15160 1376  940 R  0.3  0.0   0:10.35 top
    1 root      20   0 19356 1572 1248 S  0.0  0.0   0:01.86 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
```

Figure 5-13 Test three: CPU usage

5.2.6 Summary

As the scenario shows, WebSphere eXtreme Scale technology allows for computational levels of over 5,000 transactions per second. These high levels of computational speed require adequate computational resources. Tests show that changes in topology of application components and cache instances do not have as much impact on the performance as does the upgrade to computational resources (the number of CPUs and memory size).

5.3 Benefits to a client

There are several benefits in using the WebSphere eXtreme Scale to implement applications for an extreme transaction processing (XTP) environment:

- Flexibility
- Scalability
- High availability
- Low cost of high-performance processing

The cache-centered application provides greatly enhanced flexibility in topology choices and in changes to topology. Any application that uses cache can be implemented in a distributed manner, where each module can be specific for just one area of data processing. The topology can be changed as necessary without the need to shut down the whole application because the cache serves as the integration point between modules. Although the cache grid holds the data, the modules of the application can be moved between servers.

The application can be scaled on demand. There might be periods of time when higher levels of transaction processing are required. In this example of a stock trading application, there might be days where higher loads of new orders are expected. For those days, the application configuration can be adjusted accordingly. This configuration does not require programmatic changes and can be done at the system administration level. This process provides an advantage over some existing high-performance systems that do not allow for so much flexibility in configuration and scalability.

The fact that the data is stored in cache helps to assure high availability of data. Various cache grid configurations allow for meeting any requirement for data availability.

The cache-centered applications can be scaled easily and at low cost. The grid cache can be instantiated on the cheapest hardware, which does not have to meet any unique requirements. Lower-cost machines can be added on demand. The demand might change frequently, such as with the example stock trading application, or it might grow gradually. In any case, there is no large investment that is needed to meet the increased demand.

5.4 Conclusion

WebSphere eXtreme Scale technology is highly suitable for extreme transaction processing. It allows XTP processing to be achieved in a low-cost hardware environment and provides scalability and flexibility to meet changing enterprise requirements.

6

Installation and configuration

Installation and configuration information that is used during the implementation of the scenarios in this book is now described. Detailed information about installation and configuration can be found in the product documentation. The detailed information also includes operating systems that are supported but were not used during the implementations described.

The following topics are described:

6.1 Installing products with IBM Installation Manager

The *IBM Installation Manager* is a common application to install, remove, or modify IBM products. Installation Manager tracks the products that it installs including selectable features and maintenance updates for products. The following list includes the multiple options for installing an IBM software product by using IBM Installation Manager:

- Wizard mode
- Silent mode
- Console mode
- Command-line mode

In *wizard mode*, wizards guide you through the steps that are required to install, modify, update, roll back, or uninstall your IBM products. Use Installation Manager to install individual software packages on your local machine or, with the *IBM Packaging Utility*, to install software for an enterprise.

In *silent mode*, the installations typically require three main steps: installing IBM Installation Manager, recording a response file, and then installing the packages silently. When you install in silent mode, the user interface is not available. The response file provides the input for installation.

Console mode is for when you cannot use the IBM Installation Manager user interface. Use Installation Manager in console mode to manage installations for those products that support console mode. In console mode, select installation options from a command line.

Command-line mode is another installation option when the user interface cannot be used. In command-line mode, the Installation Manager uses the installation command line, **`imcl`**. All of the parameters and options are specified on the command line.

For the purposes of this book, IBM Installation Manager Version 1.5.1 was used. For more information, see the IBM Installation Manager V1R5 information center at this website:

http://pic.dhe.ibm.com/infocenter/install/v1r5/index.jsp

6.2 Planning and installing IBM WebSphere eXtreme Scale

In all of the scenarios that are described, IBM WebSphere eXtreme Scale was installed on Red Hat Enterprise Linux 6.0. A complete list of supported operating systems and hardware requirements can be found at the IBM WebSphere eXtreme Scale System Requirements page at this website:

http://www-01.ibm.com/software/webservers/appserv/extremescale/sysreqs/index.html

6.2.1 Planning the installation

Each type of cache topology has advantages and disadvantages. The caching topology you implement depends on the requirements of your environment and application. It is important to do appropriate planning before you start the installation for optimal performance. There are also several data capacity considerations, such as the number of systems and processors, number of partitions, and number of replicas. You can install WebSphere eXtreme Scale in a stand-alone environment or integrated with WebSphere Application Server. The topology of various components, such as catalog servers and container servers, can affect performance.

These planning considerations can be found in the Planning Overview section of the WebSphere eXtreme Scale Version 8.5 information center at this address:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fcxsovplan.html

After you determine the purposes and requirements of your deployment through the planning step, install WebSphere eXtreme Scale on your system. Detailed installation instructions and the available options can be found in the Installing section of the WebSphere eXtreme Scale Version 8.5 information center at this address:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Ftxsinstalling.html

6.2.2 Installing IBM WebSphere eXtreme Scale

IBM WebSphere eXtreme Scale is the caching solution that is used in this book. To install IBM eXtreme Scale on Red Hat Enterprise Linux, complete the following steps:

1. Start the IBM Installation Manager user interface (Figure 6-1).

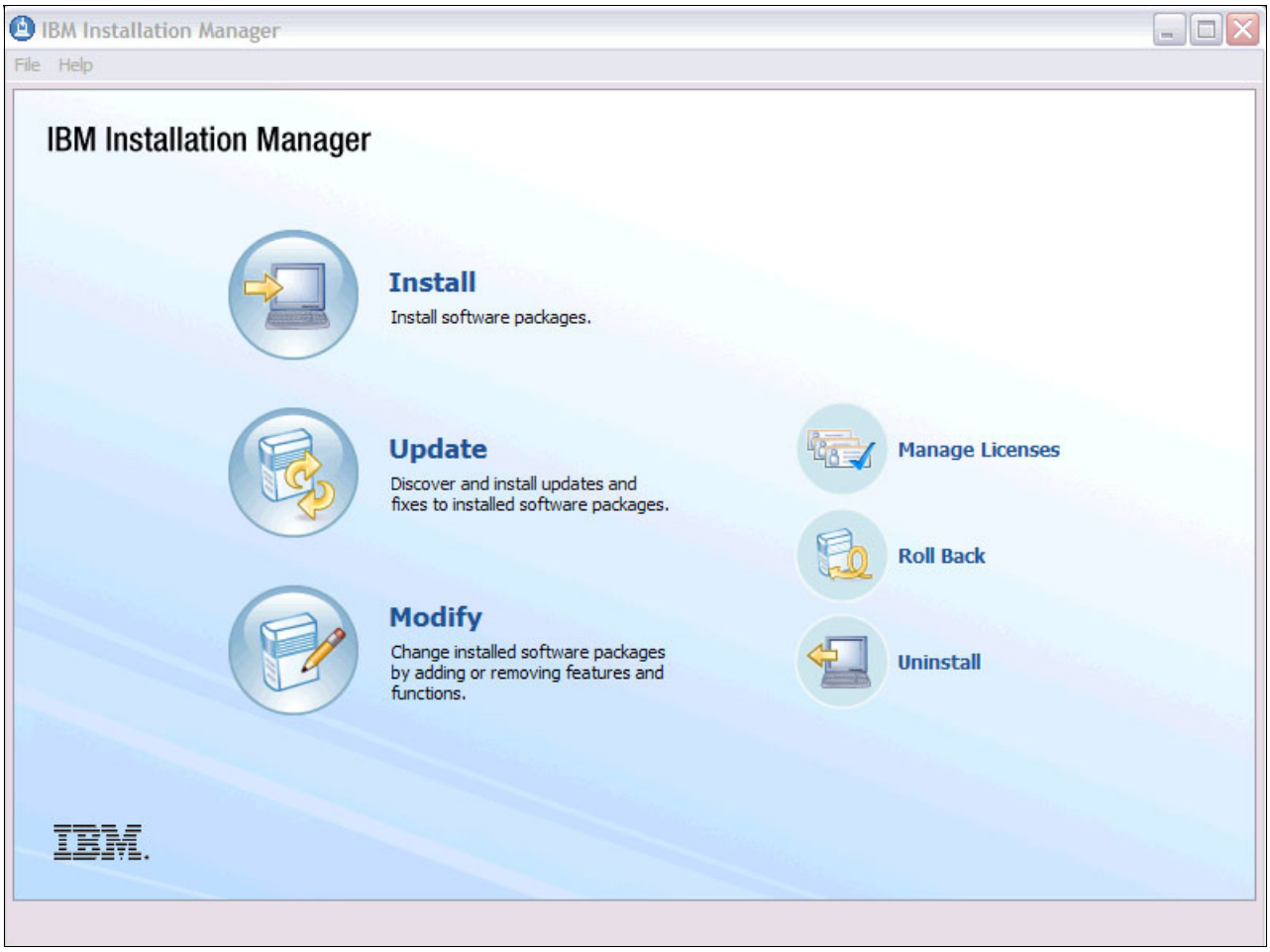

Figure 6-1 IBM Installation Manager main panel

2. Designate the location of the installation repository that contains the software packages for the product or products to be installed. To do this designation, follow these steps:
   a. Select **File** → **Preferences** to display the Preferences window (Figure 6-2).

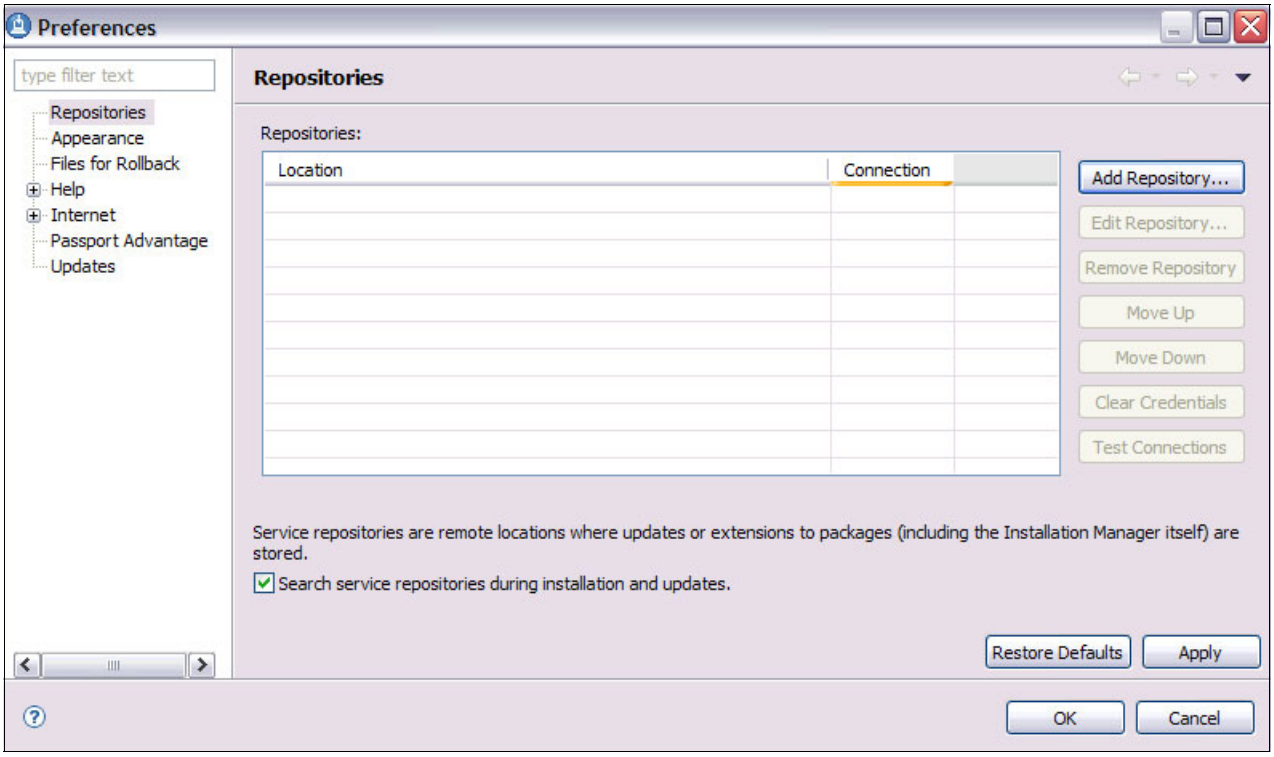

Figure 6-2 Add Repository by using the Preferences window

   b. Click **Add Repository** to add a repository location (either the physical media or the location of the downloaded product installation repository.) If you have connectivity to IBM and want to install any published fixes at the same time, select **Search service repositories during installation and updates**.
   c. Click **OK** to add the repository and return to the main IBM Installation Manager panel.
3. On the main IBM Installation Manager window, click **Install**.

4. Select the appropriate installation package, which in this example is IBM WebSphere eXtreme Scale in a stand-alone environment (see Figure 6-3). Then, click **Next**.

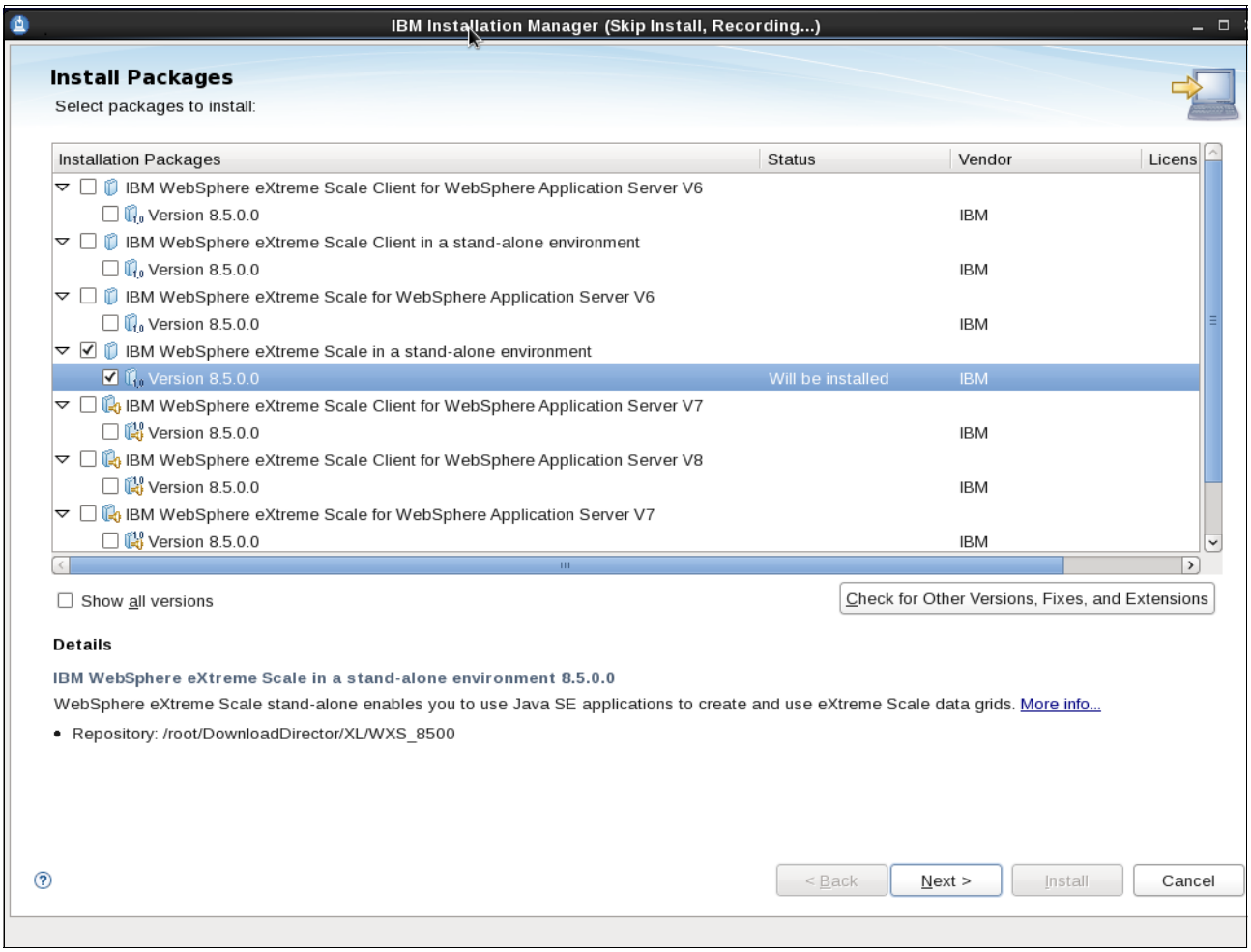

Figure 6-3 IBM WebSphere eXtreme Scale server editions

5. Proceed through the installation wizard by accepting the default values, providing the appropriate Installation Directory and Architecture, and accepting the license information. Accept the information until the Feature list is displayed.

6. Select **Server**, **Client**, and **Console** features (Figure 6-4), and then click **Next**.

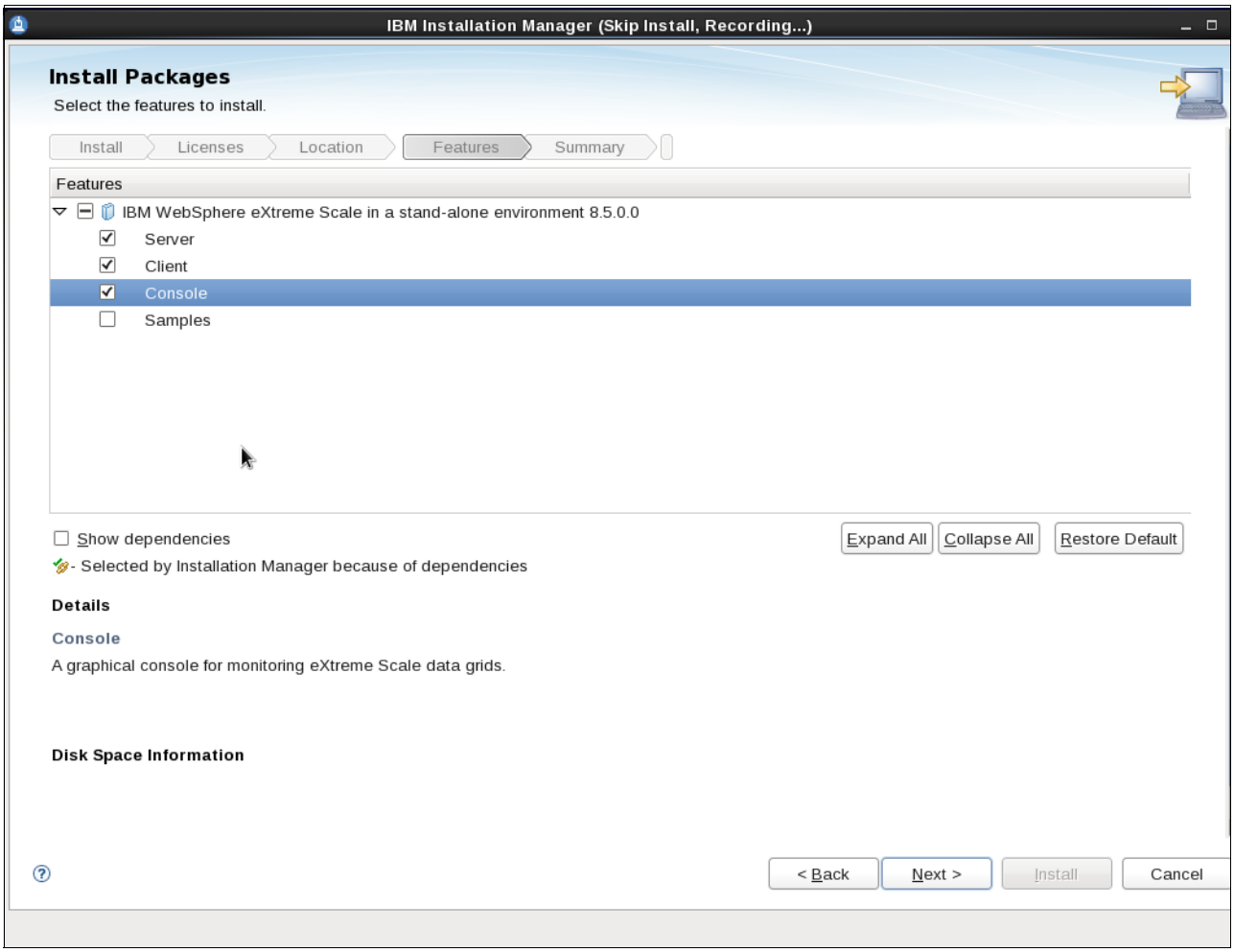

Figure 6-4 Selecting the features to install

7. Review the information and then click **Install** to start the installation procedure.
8. When installation is completed, the confirmation panel is displayed. Click **Finish**.

For information about installing IBM WebSphere eXtreme Scale on other platforms or to troubleshoot installation errors, refer to the IBM WebSphere eXtreme Scale V8.5 information center at this address:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fwelcome%2Fwelcome_xs.html

6.2.3 Understanding WebSphere eXtreme Scale parameters

WebSphere eXtreme Scale provides a wide range of settings and parameters, which allow the user to tune the cache behavior and tailor it to individual needs. When using WebSphere eXtreme Scale as a stand-alone server or as a client, the configuration of the cache parameters is achieved by using two XML files (`objectgrid.xml` and `deployment.xml`). There are examples that are in the `<installpath>/ObjectGrid/gettingstarted/xml/` directory. The following section explains the configuration parameters that can be modified and how to determine what values to use for them.

ObjectGrid descriptor file

The main XML configuration file is the `ObjectGrid` descriptor file, which contains the basic definition of the logical structure of the cache. This file is configured both on the server and the client. An example of the file that is shipped with the product is shown in Example 6-1.

Example 6-1 Example of ObjectGrid descriptor XML file

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.c Example 6-2 om/ws/objectgrid/config
../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

  <objectGrids>
    <objectGrid name="Grid">
      <backingMap name="Map1" />
      <backingMap name="Map2" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

There are two main elements in the file: `objectGrid` and `backingMap`. The `objectGrid` serves as a container for the `backingMaps`. There can be multiple `objectGrids` defined and each `objectGrid` can contain one or more `backingMaps`.

objectGrid attributes

The highest element in the cache configuration hierarchy is `objectGrid` and its attributes allow you to define the security environment and transaction timeout for all `backingMaps` contained within the grid. The list of all `objectGrid` attributes is shown in Example 6-2.

Example 6-2 objectGrid attributes and their values

```
name="objectGridName"
securityEnabled="true" | "false"
authorizationMechanism="AUTHORIZATION_MECHANISM_JASS" |
"AUTHORIZATION_MECHANISM_CUSTOM"
permissionCheckPeriod="permission_check_period"
txTimeout="seconds"
entityMetadataXMLFile="URL"
```

Transaction timeout can be customized with the help of the `txTimeout` attribute. The attribute indicates the timeout in seconds for all transactions that are performed on maps within a grid.

backingMap attributes

A `backingMap` represents the Java HashMap implementation of a single object map and has a number of attributes that define various aspects of map behavior. The attributes and their values are presented in Example 6-3.

Example 6-3 backingMap attributes and their values

```
name="objectGridName"
readOnly="true" | "false"
template="true" | "false"
pluginCollectionRef="reference to backingMapPluginCollection"
numberOfBuckets="number of buckets"
preloadMode="true" | "false"
lockStrategy="OPTIMISTIC" | "PESSIMISTIC" | "NONE"
```

```
numberOfLockBuckets="number of lock buckets"
lockTimeout="lock timeout"
copyMode="COPY_ON_READ_AND_COMMIT" | "COPY_ON_READ" | "COPY_ON_WRITE" | “NO_COPY"
| "COPY_TO_BYTES"
valueInterfaceClassName="value interface class name"
copyKey="true" | "false"
nullValuesSupported="true" | "false"
ttlEvictorType="CREATION_TIME" | "LAST_ACCESS_TIME" | "LAST_UPDATE_TIME" | NONE"
timeToLive="time to live"
streamRef="reference to a stream"
viewRef="reference to a view"
writeBehind="write-behind parameters"
evictionTriggers="MEMORY_USAGE_THRESHOLD"
```

There are situations where the cache is used to buffer static data or there is a need to prevent the data from being modified. In these situations, the performance of accessing data in a map can be improved by setting the map to be read only by using the `readOnly` attribute. By default, access to each map is set to be read/write.

All maps must be defined in the `ObjectGrid` descriptor file before the grid is instantiated. In some situations, it is difficult to determine in advance how many maps are needed by the application or applications using the cache. To eliminate the requirement of predefining all possible maps that might be needed, a common map template can be defined. This map template can be used at the run time to dynamically create maps on demand.

The map access performance can be improved by setting a higher number of buckets for the map. Increasing to a higher number of buckets also improves concurrent access to the map. For situations when data changes at a high rate, use of the backing map instead of the local client cache is advisable. To ensure the use of the backing map instead of the local client cache, set the `numberOfBuckets` attribute to `0` for each client in the `ObjectGrid` descriptor file for the client. The numberOfBuckets attribute indicates the number of hash buckets to be used by the BackingMap and is useful in fine-tuning for performance. If many entries exist, then extra buckets can improve performance. For in-line caching scenarios, where WebSphere eXtreme Scale cache is integrated with a database, a loader plug-in is defined for a map. By default, the map is not activated before the loader completes preinstalling of the map data. This setting can be configured through the `preloadMode` attribute. By setting the `preloadMode` attribute to `TRUE`, the map data is asynchronously preinstalled. This setting assures that the preinstalling of the map does not block access to the map. Additionally, writing back to the database can be automated by using the `writeBehind` attribute. This attribute defines the maximum time and maximum number of updates between automatic database updates. This mechanism is important for data integrity. Choose the settings based on data change dynamics and number of data consumers.

The data locking strategy can also be defined for a map including the type of locking, locking timeout, and the number of locking buckets. *OPTIMISTIC* locking is used when the WebSphere eXtreme Scale internal LockManager is chosen to control the map entry lock for each map entry access until the transaction accessing the entry commits. This locking strategy is normally used on data that changes infrequently. If there are frequent collisions between transactions that are updating map entries, *PESSIMISTIC* locking can be used. The *NONE* locking strategy indicates that the internal LockManager is not used and the locking control is done by the client application that uses the cache.

The copy mode defines what data value instance the client application receives when it gets an entry from the map. The instance can be the original data object stored in the map, a copy of the object, or a proxy of the object. Various settings are desirable for different scenarios

and each has different impacts on performance. By default, an application works on a copy of the object that is contained in the map as the `copyMode` defaults to `COPY_ON_READ_AND_COMMIT`. To improve performance, the `copyMode` can be set to `COPY_ON_READ`, which eliminates the copying of the object at commit time. To even further improve performance, `COPY_ON_WRITE` can be used, which eliminates copying the object and makes a proxy of the object to be received by the client application. In this case, if the objects are not modified, copying of objects does not occur. For situations where the data is not modified by a client application, setting the copy mode to `NO_COPY` provides the highest performance as copying of objects is eliminated. When complex objects are stored in a map, `COPY_TO_BYTES` can be used to improve memory footprint. To define the copy strategy for data keys to reuse the same key object for multiple map entries, set the `copyKey` attribute to `true`.

Which copy mode you choose depends on the performance requirements and the way that data held in the map is used by an application. The following sequence of questions and answers can help you decide which copy mode to use for a specific map.

Question 1: Is the high performance access to data in the map critical to my application?

NO: The copy mode can be changed to improve memory footprint.
Go to question 2.
YES: The copy mode can be changed to improve performance.
Go to question 3.

Question 2: Is minimizing the memory footprint of my map critical?

NO: Keep the default copy mode. Do not go through any more questions.
YES: If the map contains complex objects, set the copy mode to `COPY_TO_BYTES`.
Do not go through any more questions.

Question 3: Will the data in the map be modified by my application?

NO: Set copy mode to `NO_COPY` to ensure the highest performance.
Do not go through any more questions.
YES: The copy mode can be changed to improve performance. Go to question 4.

Question 4: Will data in the map be modified often by my application?

NO: Set the copy mode to `COPY_ON_READ`.
YES: Set the copy mode to `COPY_ON_WRITE`.

When you get a value from a grid, a *null* can have different meanings: it can be a null value or it can mean that the value does not exist in the map. By default, `null` means that the value does not exist in the map. To change this default behavior, set the `nullValuesSupported` attribute to `false`.

Eviction strategy can be defined individually for each map. This strategy allows removing the data lifecycle control from the application that uses the cache. In many cases data has a precisely defined lifespan. For example, a cache can hold product information for a discount sale with the duration set for a limited time, such as 48 hours. The cache can be preinstalled, at the beginning of the sale period, with the discounted product pricing information and the eviction strategy set to eliminate the data from the cache in 48 hours after creation. The eviction strategy can be defined by adjusting the `timeToLive` setting. This setting indicates how many milliseconds, after a point in time that is defined by the `ttlEvictorType` attribute, that each entry in the map is to be eliminated. The eviction strategy can also be set in more generic ways by setting *eviction triggers*. Currently, there is only one trigger defined that activates eviction in cases where the map reaches the maximum memory usage threshold. For this eviction to take effect, both the `memoryThresholdPercentage` property has to be set in the `server.properties` file and the `evictionTriggers="MEMORY_USAGE_THRESHOLD"` attribute has to be set in the backingMap.

Deployment policy file

For any cache configuration that spans multiple Java virtual machines (JVMs), the deployment policy XML file (`deployment.xml`) is required in addition to the `ObjectGrid` descriptor file. One of the main purposes of the deployment file is to define the partitioning and replication of object maps. The example deployment file that is shipped with WebSphere eXtreme Scale is shown in Example 6-4. There is one `objectgridDeployment` element for each grid in the cache. Each `objectgridDeployment` element contains one or more `mapSet` elements. Each `mapSet` element contains one or more references to object maps.

Example 6-4 Example of deployment policy XML file

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0"
maxSyncReplicas="1" >
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

mapSet attributes

The `mapSet` groups object maps and defines a common placement and replication strategy for all maps contained in the `mapSet`. The list of attributes that can be set on a `mapSet` is presented in Example 6-5.

Example 6-5 mapSet attributes and their values

```
name="mapSetName"
numberOfPartitions="numberOfPartitions"
minSyncReplicas="minimumNumber"
maxSyncReplicas="maximumNumber"
maxAsyncReplicas="maximumNumber"
replicaReadEnabled="true|false"
numInitialContainers="numberOfInitialContainersBeforePlacement"
autoReplaceLostShards="true|false"
developmentMode="true|false"
placementStrategy="FIXED_PARTITIONS | PER_CONTAINER"
```

The number of partitions over which a map is distributed must be defined. For this attribute to make sense, its value will not exceed the number of container servers in the cache infrastructure less one. Distributing each map over a higher number of partitions improves performance when rebalancing shards during a container server failure. Smaller partitions are easier to move around and to duplicate. Keep in mind this fact as you balance the needs of your system.

The number of replicas must be defined. All maps in the map set can have a different number of synchronous and asynchronous replicas. The replicas usually are not accessible by client applications. However, in some cases, for performance improvement reasons, it is desirable for the replicas to be reading accessible. Reading can be enabled with the help of the

`replicaReadEnabled` attribute. When reading from replicas is enabled, the client application must account for the potential of receiving stale data.

During the start of a multi-container WebSphere eXtreme Scale cache, every new container server startup starts a shard placement process. Whenever extra container servers start and register with the existing catalog domain, the placement process rebalances shard distribution. For environments where computational resources must be controlled, the placement process execution can be delayed to run after a minimum number of container servers start. The delay is defined by setting the value of the `numInitialContainers` attribute.

Whenever a server container shuts down, all partitions with primary shards on this server get rebalanced. One of the replicas in the partition is promoted to a primary and, as default, a new replica is created on a different container. If for some reason the new replica is not created, check the `autoReplaceLostShards` attribute and set to `false`.

In some cases, multiple shards of the same partition can exist on the same host. This scenario can happen in a development environment or where speed of data access is critical.

The placement strategy can be defined so that all primary shards are distributed across all container servers. For this option, the `placementStrategy` attribute is set to `FIXED_PARTITIONS`. To activate the placement of all primary shards on each container server, set the `placementStrategy` attribute to `PER_CONTAINER`.

After the XML configuration for WebSphere eXtreme Scale 8.5 is complete, start or restart the server and verify that the server started successfully. For more information about how to do the startup, see the IBM WebSphere eXtreme Scale Version 8.5 information center at this address:

http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp

6.2.4 Running WebSphere eXtreme Scale servers on Liberty profile servers

This section describes the main steps to configure WebSphere eXtreme Scale to run by using a WebSphere Application Server V8.5 Liberty profile server.

Creating the Liberty profile server

For WebSphere eXtreme Scale, you must have both catalog servers and container servers. In this example, we use the WebSphere Application Server V8.5 Liberty profile servers as the underlying application server for the catalog and container servers.

To create the topology for the mobile scenario, which is shown in Figure 3-7 on page 32, create three catalog servers and four container servers, all on different machines. You must have already installed IBM WebSphere eXtreme Scale on each of these servers, as detailed in 6.2, “Planning and installing IBM WebSphere eXtreme Scale” on page 104.

In the steps that follow, the server names and the host name of the machine that they are running on are listed in Table 6-1.

Table 6-1 Server name, host name, and server types

Liberty profile server name	Host name	Type of server
catalog1	catalogHost1	catalog server
catalog2	catalogHost2	catalog server
catalog3	catalogHost3	catalog server

Liberty profile server name	Host name	Type of server
container1	containerHost1	container server
container2	containerHost2	container server
container3	containerHost3	container server
container4	containerHost4	container server

To create a Liberty profile server, run the command **server create <server_name>** from *<WLP_INSTALL_DIR>*/bin, where *<server_name>* is the Liberty profile server name from the preceding table. This name is based on the host name of the machine in which you are running the command.

Create all of the catalog and container servers for the topology in the same way.

Configuring the catalog servers

To run the WebSphere eXtreme Scale catalog server in a Liberty profile, you must enable and configure WebSphere eXtreme Scale for the Liberty Profile. To do this process for each of the three catalog servers that you created, follow these steps:

1. Edit the server.xml file for the catalog server, as shown in Example 6-6. The file is in the *<WLP_USER_DIR>*/servers/*<server_name>* directory, where *<WLP_USER_DIR>* is the user directory for the Liberty profile and *<server_name>* is the Liberty profile server name.
   - Change the server description.
   - To enable the feature, add the feature **eXtremeScale.server-1.0** to the featureManager.
   - Specify the file path to the properties file in a serverProps attribute inside the com.ibm.ws.xs.server.config element.
   - Add the <xsServer> line, which defines this WebSphere eXtreme Scale installation as a catalog server and the listenerPort.

   Example 6-6 server.xml for catalog server

   ```
   <server description="catalog server">
   <!-- Enable features -->
       <featureManager>
   <feature>eXtremeScale.server-1.0</feature>
   </featureManager>
   <com.ibm.ws.xs.server.config serverProps="server.properties"/>
   <xsServer isCatalog="true" listenerPort="2809"/>
   </server>
   ```

2. Create a file named server.properties in the same directory as the server.xml. This file is used to define extra properties for the server during startup and is referenced in the server.xml file that you just edited. Add the line that is shown in Example 6-7. When the catalog server is started, the catalog service endpoints must be specified. Because we are using the Liberty Profile, the endpoints are specified in the server.properties file.

   Example 6-7 New server.properties file for each catalog server

   ```
   catalogClusterEndPoints=catalog1:catalogHost1:6601:6602,catalog2:catalogHost2:6
   601:6602,catalog3:catalogHost3:6601:6602
   ```

Configuring the container servers

Enable and configure the WebSphere eXtreme Scale container servers for the Liberty Profile by following these steps for each of the four container servers:

1. Edit the `server.xml` file for the container server to match the XML shown in Example 6-8. The file is located the *<WLP_USER_DIR>*/servers/*<server_name>* directory, where *<WLP_USER_DIR>* is the user directory for the Liberty profile and *<server_name>* is the Liberty profile server name.
   - Change the server description.
   - To enable the feature, add the feature **eXtremeScale.server-1.0** to the featureManager.
   - Specify the file path to the properties file in a serverProps attribute inside the com.ibm.ws.xs.server.config element.

   Example 6-8 server.xml for container server

```
<server description="grid container">
      <!-- Enable features -->
    <featureManager>
        <feature>eXtremeScale.server-1.0</feature>
    </featureManager>
      <!-- WXS settings-->
    <com.ibm.ws.xs.server.config serverProps="server.properties"/>
</server>
```

2. Create a file named `server.properties` in the same directory as the server.xml. This file is used to define extra properties for the server during startup and is referenced in the `server.xml` file that you just edited. Add the line that is shown in Example 6-9. The **catalogServiceEndPoints** property defines the endpoints of the catalog domain.

   Example 6-9 New server.properties for each container server

```
catalogServiceEndPoints=catalogHost1:2809,catalogHost2:2809,catalogHost3:2809
```

3. Create a directory that is called `grids` under the *<WLP_USER_DIR>*/servers/*<server_name>* directory where *<WLP_USER_DIR>* is the user directory for the Liberty profile and *<server_name>* is the Liberty profile server name. In this file, create two new files, the objectGrid descriptor file, `objectgrid.xml`; and the deployment file, `objectGridDeployment.xml`.

   Example 6-10 shows the contents of the `objectgrid.xml` file that is used in this scenario. In this file, we define a single data grid, named *Grid*, and two maps, named *Map1* and *Map2*.

   Example 6-10 Contents of objectgrid.xml file

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

    <objectGrids>
        <objectGrid name="Grid">
            <backingMap name="Map1" />
            <backingMap name="Map2" />
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Example 6-11shows the `objectGridDeployment.xml` file that is used in this scenario. The file defines the map set parameters, including that the map set for this scenario has four partitions and an initial set of two containers.

Example 6-11 Contents of objectGridDeployment.xml file

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="4" minSyncReplicas="0"
maxSyncReplicas="1" numInitialContainers="2">
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

The `grids` directory is monitored by WebSphere eXtreme Scale. When a new `objectgrid.xml`, `objectGridDeployment.xml`, or both files are found, WebSphere eXtreme Scale creates and starts a new container that corresponds to the definitions in those files. When one of these files is deleted, WebSphere eXtreme Scale stops the corresponding container. When the files are modified, WebSphere eXtreme Scale stops and restarts the container. Multiple containers can exist in the same WebSphere eXtreme Scale server, which requires that subdirectories exist inside the `grids` directory.

Starting WebSphere eXtreme Scale servers

There are several steps that are needed to start the WebSphere eXtreme Scale servers and do basic verification that the servers are running properly.

Object Request Broker settings

The WebSphere eXtreme Scale run time depends highly on the Object Request Broker (ORB) run time. The following dependencies are essential when you run WebSphere eXtreme Scale on the Liberty server:

- Java runtime environment (JRE): IBM JRE or other JRE
- ORB vendor: IBM or other vendor
- ORB settings: Prepare an `orb.properties` file in `JAVA_HOME/jre/lib`. Example 6-12 shows the contents of an example `orb.properties` file.

Example 6-12 Example of orb.properties

```
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.Obj
ectGridInitializer

com.ibm.CORBA.FragmentSize=0
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10
```

Verifying that all servers are started successfully

You can use the following methods to verify whether the WebSphere eXtreme Scale environment was started successfully:

- Verify that the Liberty server is started successfully from the SystemOut log.

  If the Liberty server is started successfully, it normally outputs the message that is shown in Example 6-13.

  Example 6-13 Example of SystemOut

  ```
  [AUDIT] CWWKF0011I: The server wlserver is ready to run a smarter planet.
  ```

- Check the Liberty server logs.

  From the `WLP_USER_DIR/servers/yourserver/logs/messages.log` file, you can find the catalog and container logs if the servers started successfully, as shown in Example 6-14.

  Example 6-14 Example of catalog server logs to indicate a successful start

  ```
  [8/24/12 12:05:05:382 GMT] 00000015 com.ibm.ws.objectgrid.ServerImpl
  I CWOBJ1001I: ObjectGrid Server catalog1 is ready to process requests.
  ```

  Example 6-15 shows an example of grid container logs that indicate a successful start.

  Example 6-15 Example of grid container logs to indicate a successful start

  ```
  [8/24/12 12:39:45:137 GMT] 00000015 com.ibm.ws.objectgrid.ServerImpl
  I CWOBJ1001I: ObjectGrid Server container1 is ready to process requests.
  ```

- Use the **`netstat`** command.

  Example 6-16 shows an output of the command with one catalog server listening on three ports.

  Example 6-16 Example of used ports for a catalog server

  ```
  #netstat -a | grep LIBSERV1
  LIBSERV1 00000EC1 127.0.0.1..2809        127.0.0.1..2060        Establsh
  LIBSERV1 00000EEA 172.17.206.250..2071   172.17.206.250..2066   Establsh
  LIBSERV1 00000E1C 127.0.0.1..2004        0.0.0.0..0             Listen
  LIBSERV1 00000E20 0.0.0.0..2809          0.0.0.0..0             Listen
  LIBSERV1 00000EDD 172.17.206.250..2809   172.17.206.250..2068   Establsh
  LIBSERV1 00000EC4 172.17.206.250..2809   172.17.206.250..2061   Establsh
  LIBSERV1 00000EDA 127.0.0.1..2809        127.0.0.1..2067        Establsh
  LIBSERV1 00000E23 0.0.0.0..1099          0.0.0.0..0             Listen
  LIBSERV1 00000ED1 172.17.206.250..2064   172.17.206.250..2059   Establsh
  ```

  In Example 6-16, `2004` is for communication between catalog servers, `2809` is the port for communication with grid client and container server, and `1099` is the JMX service port.

- Use the **`xscmd`** utility to check the `map/grid/container` data layout.

  Check the map size by using the following command:

  ```
  ./xscmd.sh -c showMapSizes -cep host1:port1,host2:port2
  ```

Example 6-17 shows the command output for objectGrid definition that is shown in Example 6-10 on page 115 and the deployment that is shown in Example 6-11 on page 116.

Example 6-17 Example of xscmd output for showMapSize

```
# ./xscmd.sh -c showMapSizes -cep localhost:2809,localhost:2810
......
CWXSI0068I: Executing command: showMapSizes

*** Displaying results for Grid data grid and mapSet map set.

*** Listing maps for container1 ***
Map Name Partition Map Entries Used Bytes Shard Type         Container
-------- --------- ----------- ---------- ----------         ---------
Map1     0         0           0          Primary            container1_C-0
Map1     1         0           0          SynchronousReplica container1_C-0
Map1     2         0           0          SynchronousReplica container1_C-0
Map1     3         0           0          Primary            container1_C-0
Map2     0         0           0          Primary            container1_C-0
Map2     1         0           0          SynchronousReplica container1_C-0
Map2     2         0           0          SynchronousReplica container1_C-0
Map2     3         0           0          Primary            container1_C-0
Server total: 0 (0 B)

*** Listing maps for container2 ***
Map Name Partition Map Entries Used Bytes Shard Type         Container
-------- --------- ----------- ---------- ----------         ---------
Map1     0         0           0          SynchronousReplica container2_C-0
Map1     1         0           0          Primary            container2_C-0
Map1     2         0           0          Primary            container2_C-0
Map1     3         0           0          SynchronousReplica container2_C-0
Map2     0         0           0          SynchronousReplica container2_C-0
Map2     1         0           0          Primary            container2_C-0
Map2     2         0           0          Primary            container2_C-0
Map2     3         0           0          SynchronousReplica container2_C-0
Server total: 0 (0 B)

Total catalog service domain count: 0 (0 B)
(The used bytes statistics are accurate only when you are using simple objects
or the COPY_TO_BYTES copy mode.)

CWXSI0040I: The showMapSizes command completed successfully.
```

To verify failover data protection, check the placement of primary shards and their distribution across all containers by using the following command:

```
./xscmd.sh -c showPlacement -cep <host1>:<port1>,<host2>:<port2>
```

6.3 Configuring IBM WebSphere DataPower XC10 Appliance

The WebSphere Data Power XC10 Appliance is designed to work at the caching tier of your environment to support simplified deployment and increased security. The following section

focuses on how to create a data grid on the WebSphere DataPower XC10 and assumes that you already completed the basic configuration of the appliance.

6.3.1 Initialization and basic configuration

Do the initialization and basic configuration of the WebSphere DataPower XC10 Appliance before running the caching configurations described in the next section.

For information about how to do the basic appliance configuration, refer to the IBM WebSphere DataPower XC10 Appliance information center available at this address:

http://publib.boulder.ibm.com/infocenter/wdpxc/v1r0/index.jsp

6.3.2 Creating a data grid on the WebSphere DataPower XC10

The simple data grid configuration that is described here is used for two different scenarios in this book. In the scenario in section 4.2, “DataPower XC10 as a cache solution for DataPower XI52” on page 46, the data grid is used in the side cache solution for DataPower XI52. And, in the scenario in section 3.3, “Integrating IBM Worklight and the XC10 appliance” on page 29, it is used as a side-cache for the IBM Worklight server.

The data grid contains two maps. The first map is used to cache the results of a web service request to obtain client information that is aggregated with the results of the customer profile database query. The second map is the default map that is created automatically along with the data grid and has the same name as the grid. When you use the DataPower XC10 REST Gateway application programming interface (API), the required maps are created dynamically based on the template that is provided in the grid access configuration file (`cache-config.xml`).

To create a data grid, complete the following steps:

1. Open the DataPower XC10 login page and enter a valid user name and password for your XC10, as shown in Figure 6-5.

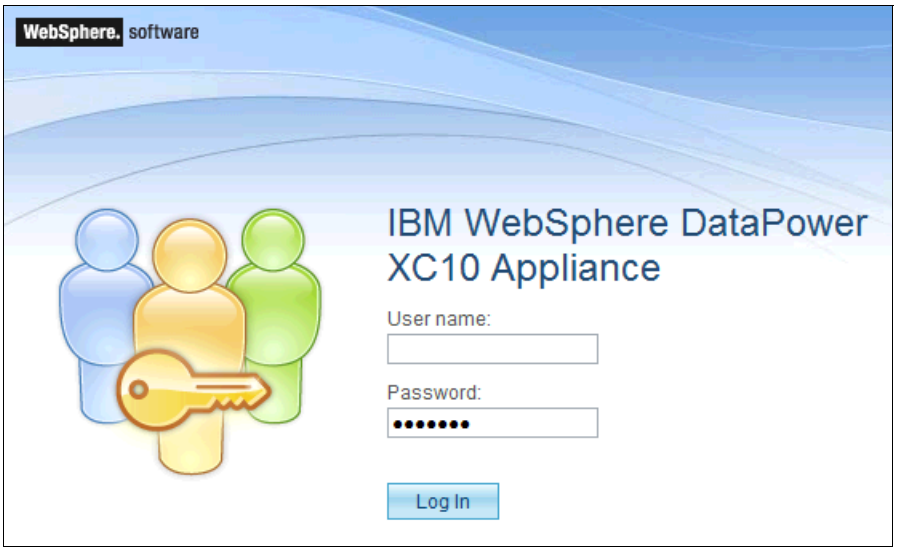

Figure 6-5 WebSphere DataPower XC10 Appliance login page

2. On the Control Panel click **Data Grid** → **Simple Data Grid**, as shown in Figure 6-6.

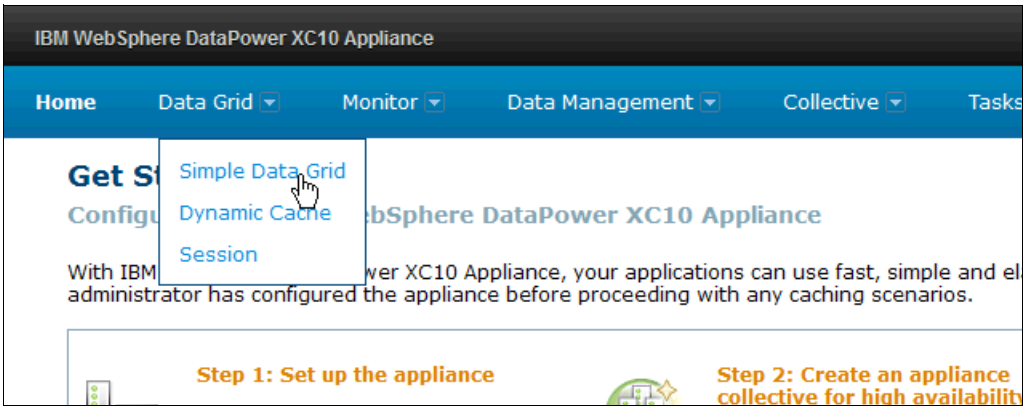

Figure 6-6 Select Simple Data Grid from the Data Grid menu

3. Add a Simple Data Grid by clicking the green plus sign icon, as show in Figure 6-7.

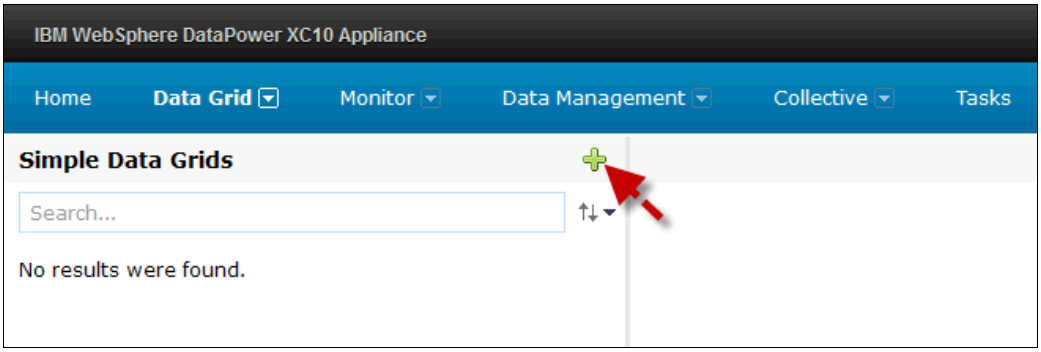

Figure 6-7 Create a new simple data grid

4. Enter a name for the data grid, then click **OK**. For the purposes of this book, the name **sw215grid** is used.

5. It can take a few minutes to do the actual data grid creation. The next message gives you the option of monitoring the data grid creation by using the tasks view, or staying on the current view until the task completes. Click **Tasks View** to monitor completion of the task by using the Task View. The Task View, which is shown in Figure 6-8, shows the log of the ongoing and completed tasks. After you refresh the page, you can see whether the simple data grid creation task completed successfully.

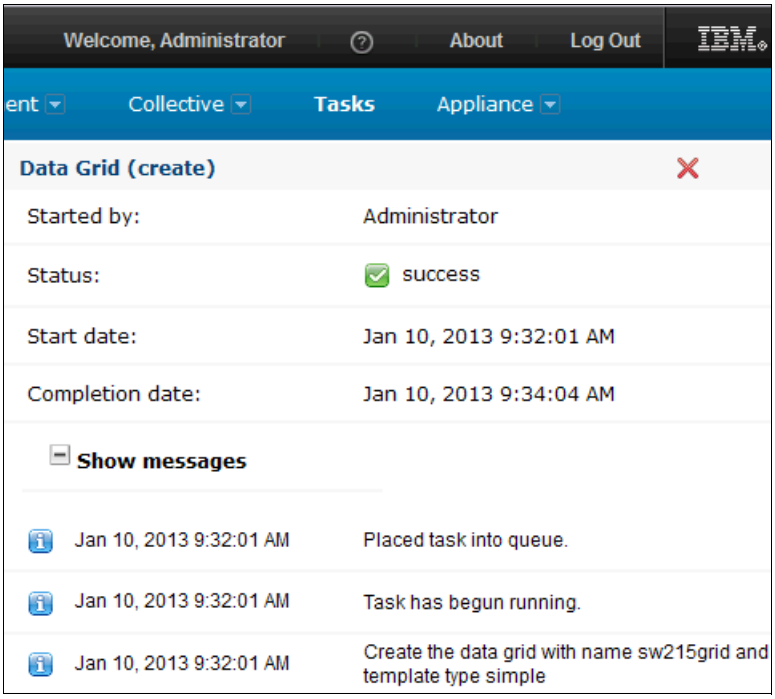

Figure 6-8 Simple data grid creation task details

6. When you create a simple grid, a default (static) map and a set of dynamic maps are automatically created. By default, there is no *time to live* (*TTL*) evictor that is configured for the default map. However, you can enable a TTL evictor policy for the default map. To adjust this TTL evictor, display the properties for the sw215grid by clicking **Data Grid** → **Simple Data Grid** and then clicking the sw215grid link.
7. Expand the **Show advanced attributes** section and set the eviction type and period. For the purposes of this book, set the evictor to be **Creation time** and set the time to be **10** seconds after that creation time, as shown in Figure 6-9.

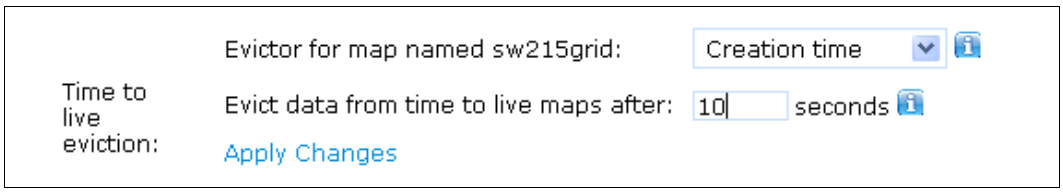

Figure 6-9 Setting time to live eviction

8. Click **Apply Changes** to save your configured values.

The WebSphere DataPower XC10 simple data grid is now ready to be used for the scenarios as described.

Follow the same steps to create a second simple data grid named **MobileGrid**.

6.3.3 Using the monitoring from the XC10 console

You can monitor the map size and container statistics from the XC10 console.

Monitoring the data grid

From the menu, select **Monitor** → **Data Grid Detail Reports** to display the page, as shown in Figure 6-10.

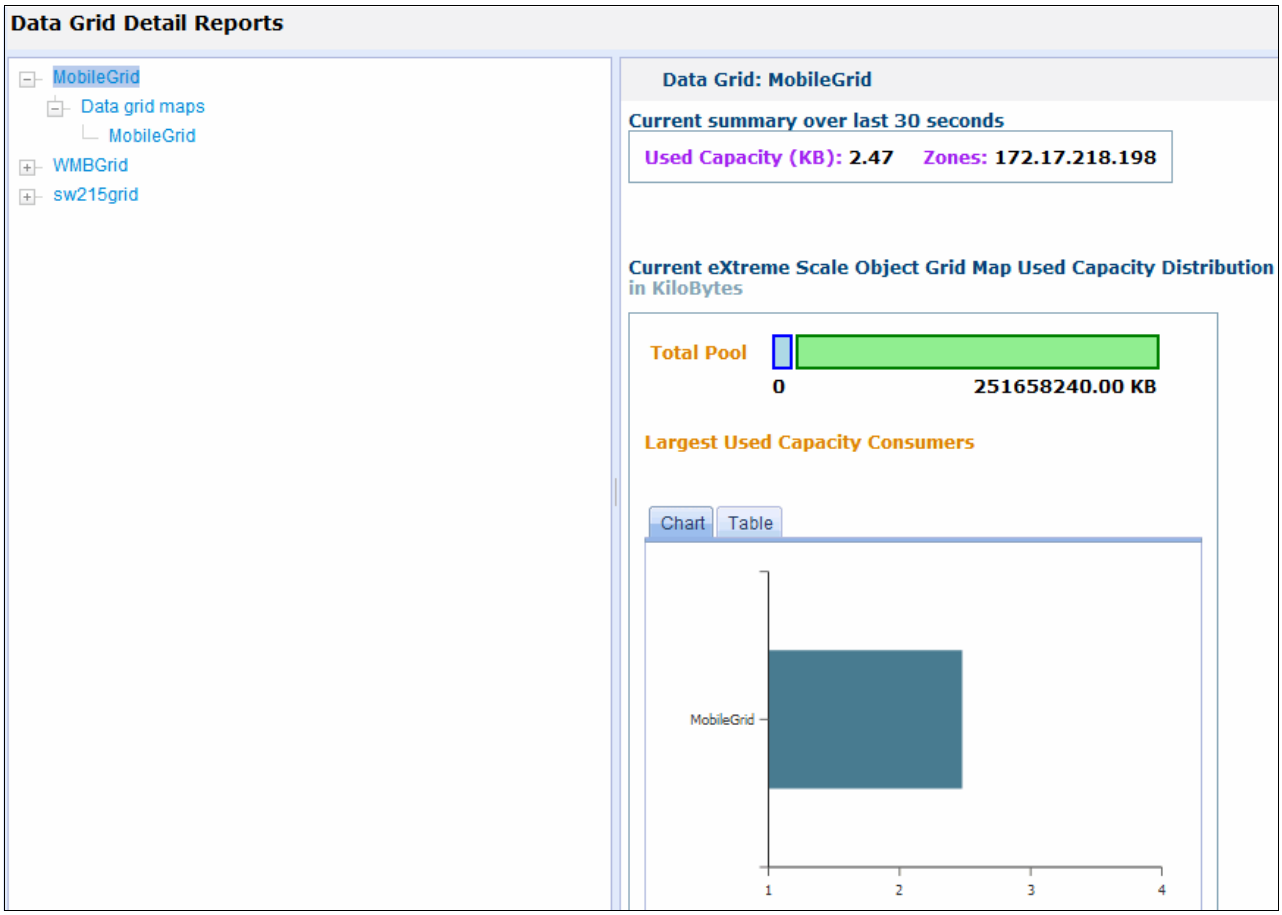

Figure 6-10 Grid details

Monitoring the map

From the menu, select **Monitor** → **Data Grid Detail Reports** to display the window that is shown in Figure 6-11.

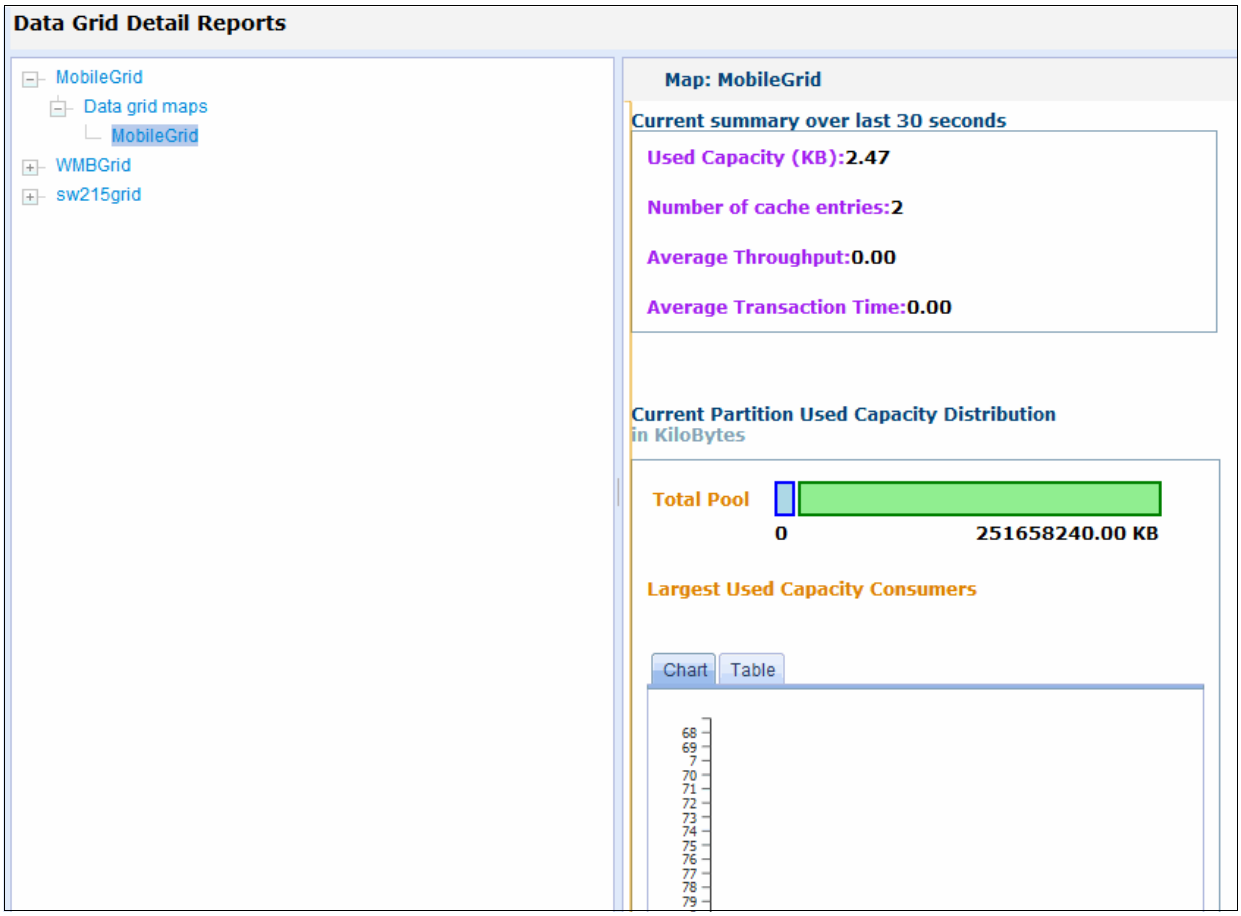

Figure 6-11 Map details

6.4 Installing IBM Worklight 5.0

Installing IBM Worklight 5.0 requires the use of IBM Installation Manager V1.5.1 or later using the graphical user interface (GUI).

The IBM Worklight V5.0 Administration Guide provides information for planning your IBM Worklight V5.0 environment and topology. IBM Worklight Server can be installed in many different network configurations. These configurations might include several DMZ layers, reverse proxies, network address translation (NAT) devices, firewalls, high availability components such as load balancers, IP sprayers, and clustering. The Administration Guide can be found at the following address:

http://public.dhe.ibm.com/ibmdl/export/pub/software/mobile-solutions/worklight/docs/wl_5_0_admin.pdf

For information about the production functionality of IBM Worklight, refer to the IBM Worklight website:

http://www-01.ibm.com/software/mobile-solutions/worklight/

For detailed system requirements, a list of the supported operating systems, and other information about Worklight Server Enterprise Edition and its installation, refer to the IBM Mobile Foundation support website:

http://www-01.ibm.com/support/docview.wss?uid=swg27024838

6.4.1 Installing IBM Worklight Server

The following installation steps describe the basic setup of IBM Worklight Server, including IBM Worklight Console and IBM Worklight Application Center, running on the IBM WebSphere Application Server V8.5 Liberty Profile.

Ensure that the installation repository for IBM Worklight Server is available, either in an accessible directory structure or on physical media. To install IBM Worklight Server, complete the following steps:

1. Start the IBM Installation Manager user interface (Figure 6-12).

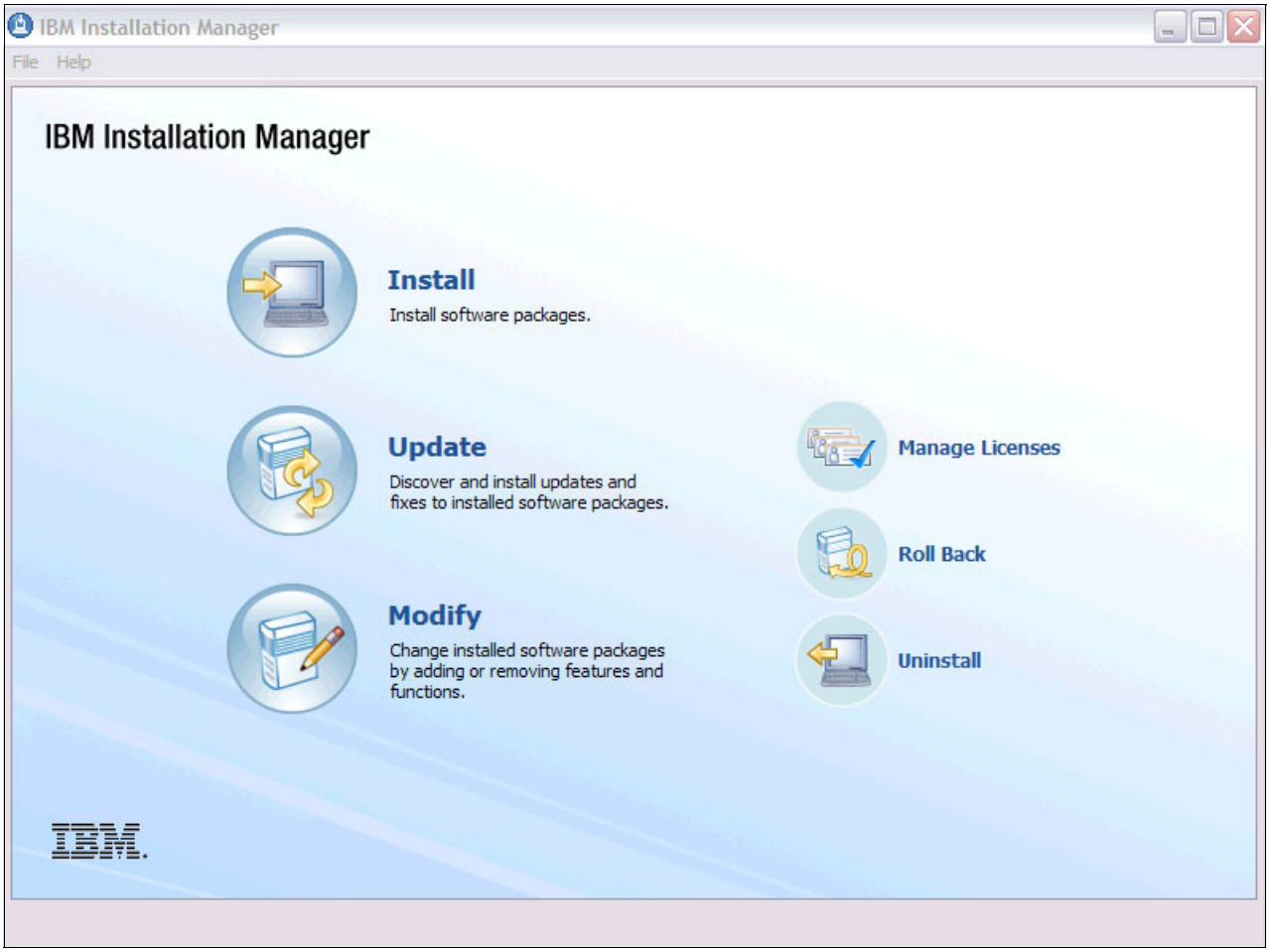

Figure 6-12 IBM Installation Manager main panel

2. Designate the location of the installation repository that contains the software packages for the product or products to be installed. To do this designation, follow these steps:

   a. Select **File** → **Preferences** to display the Preferences window (Figure 6-13).

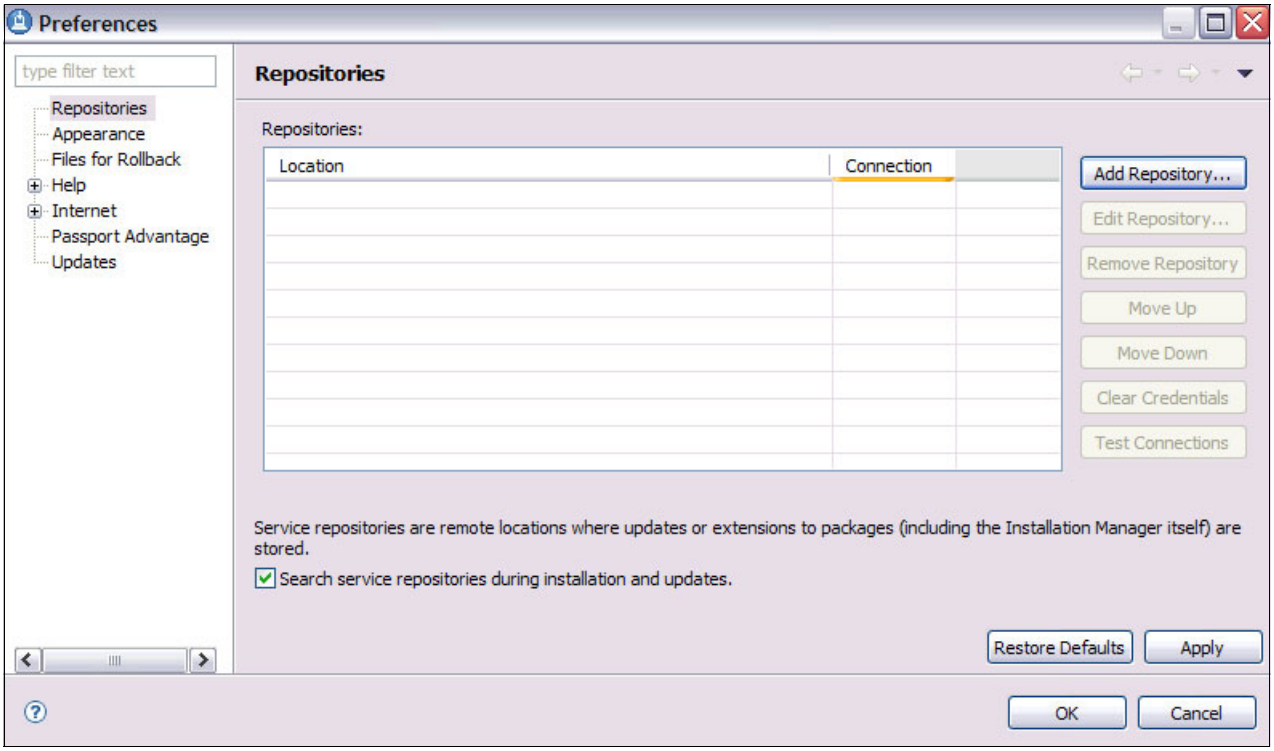

Figure 6-13 Add Repository using the Preferences window

   b. Click **Add Repository** to add a repository location (either the physical media or the location of the downloaded product installation repository.) If you have connectivity to IBM and want to install any published fixes at the same time, select **Search service repositories during installation and updates**.

3. Click **OK** to add the repository and return to the main IBM Installation Manager panel (Figure 6-1 on page 105).

4. Click **Install**.

5. Select the appropriate installation package, for example, IBM Worklight Server for IBM Worklight Enterprise Edition Version 5.0.0. See Figure 6-14.

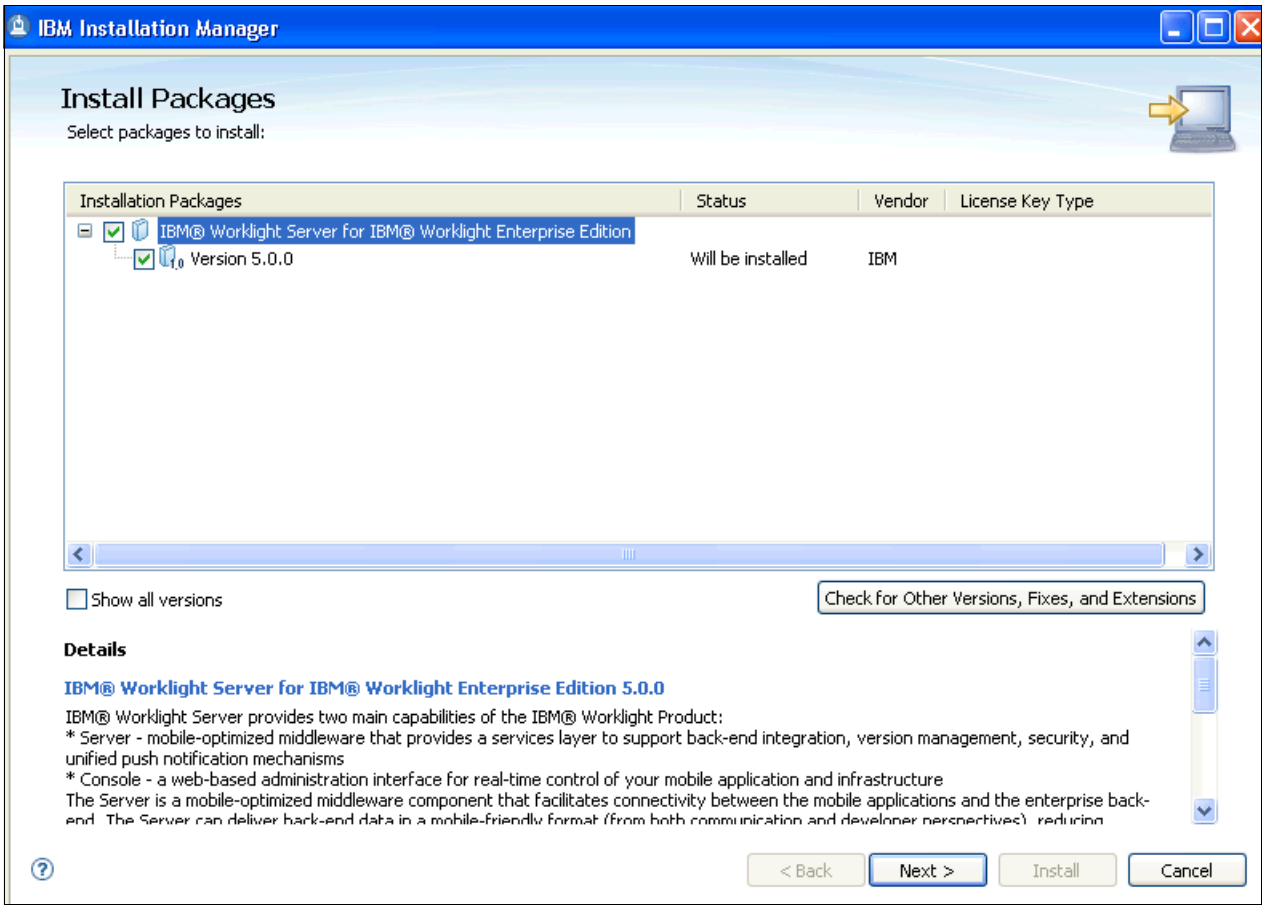

Figure 6-14 Selecting the installation packages for IBM Worklight Server V5.0

6. Click **Next** and continue through the installation wizard, accepting the default options, until you reach the Database panel.

7. Define the database that IBM Worklight uses. In this example, it is Apache Derby 10.5, as shown in Figure 6-15. Then, click **Next**.

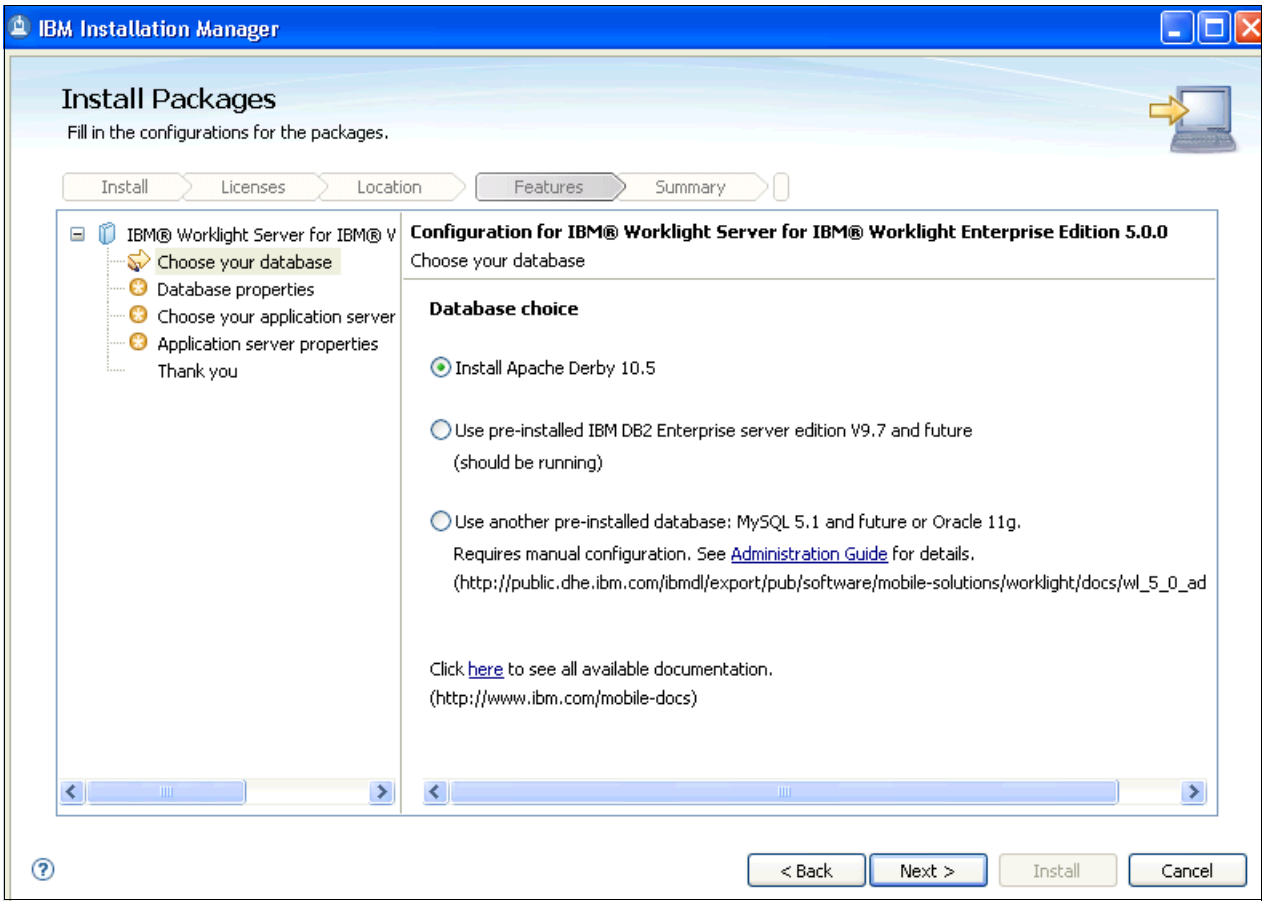

Figure 6-15 Selecting the database to be used

8. The selection of Apache Derby means that no additional parameters are required and the databases are created automatically. If a different database management system is selected, more parameters and steps might be required. Click **Next**.

9. Define the application server that IBM Worklight uses. In this example, it is WebSphere Application Server Liberty profile, as shown in Figure 6-16. Then, click **Next**.

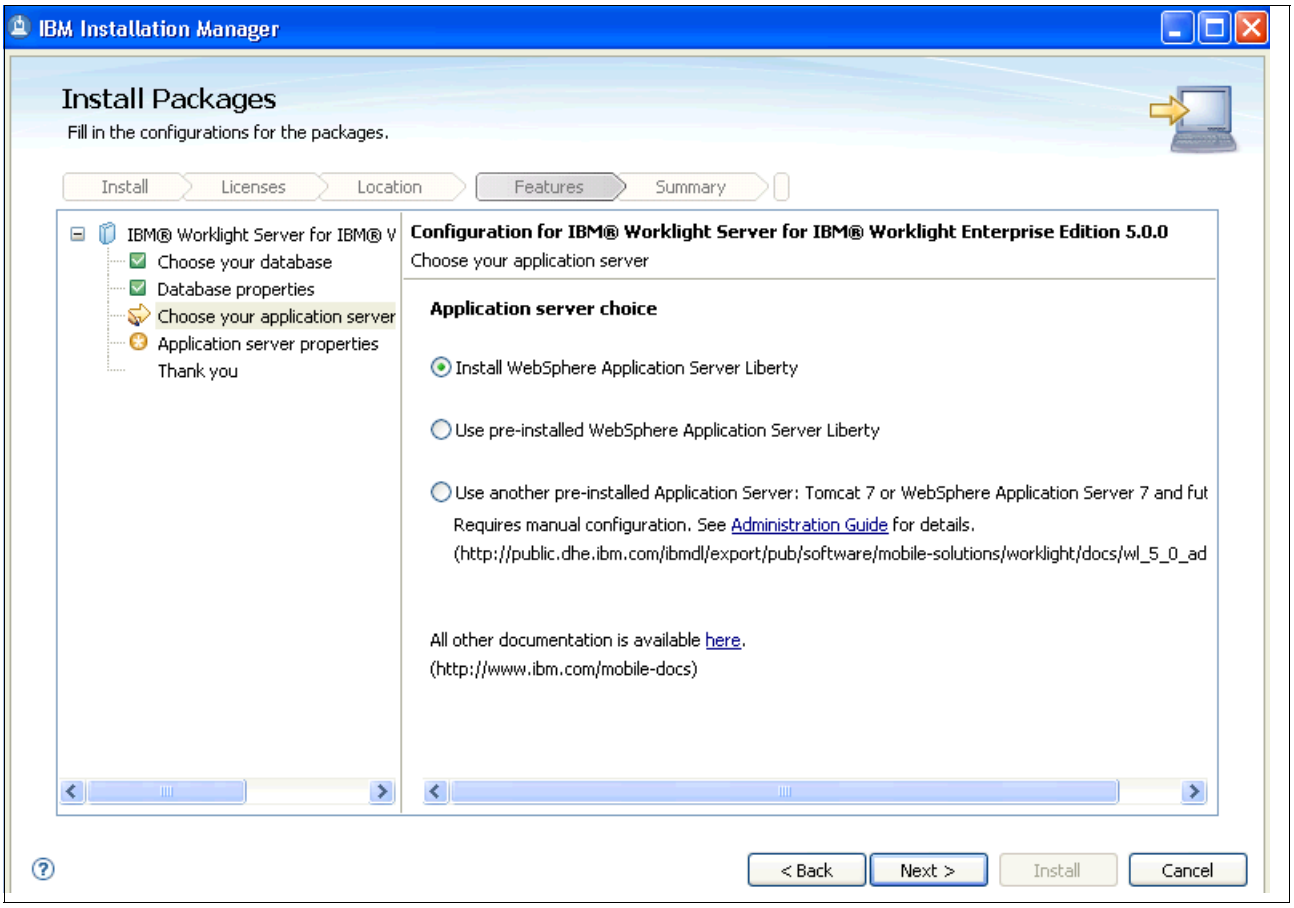

Figure 6-16 Selecting the application server

10. The selection of WebSphere Application Server Liberty profile means that no additional parameters are required and the application server is created and configured automatically. If a different application server is selected, more parameters and steps might be required. Click **Next**.
11. Click **Next** on the Thank you panel and then click **Install** to start the installation procedure.
12. When installation is completed, the confirmation panel is displayed. Click **Finish**.

After you complete the basic installation of IBM Worklight, no additional configuration is required. The remaining task is to install the applications as described in section 3.2.2, “Running IBM Worklight server on a Liberty profile” on page 26.

6.4.2 Installing IBM Worklight Studio

IBM Worklight Studio is used to develop the mobile application. For the purposes of this book, IBM Worklight Studio is installed on Mac OS X by using the “Install new software” feature of Eclipse. Before you start this installation, ensure that you have the following prerequisites:

- An existing, supported level of Eclipse
- Access to the IBM Worklight Studio V5.0 Update Site Archive file
- Oracle Java Runtime Environment Version 1.6 or higher

To install IBM Worklight Studio, follow these steps:

1. Start your existing Eclipse integrated development environment (IDE) and then click **Help** → **Install New Software**, which displays the Available Software window. See Figure 6-17.

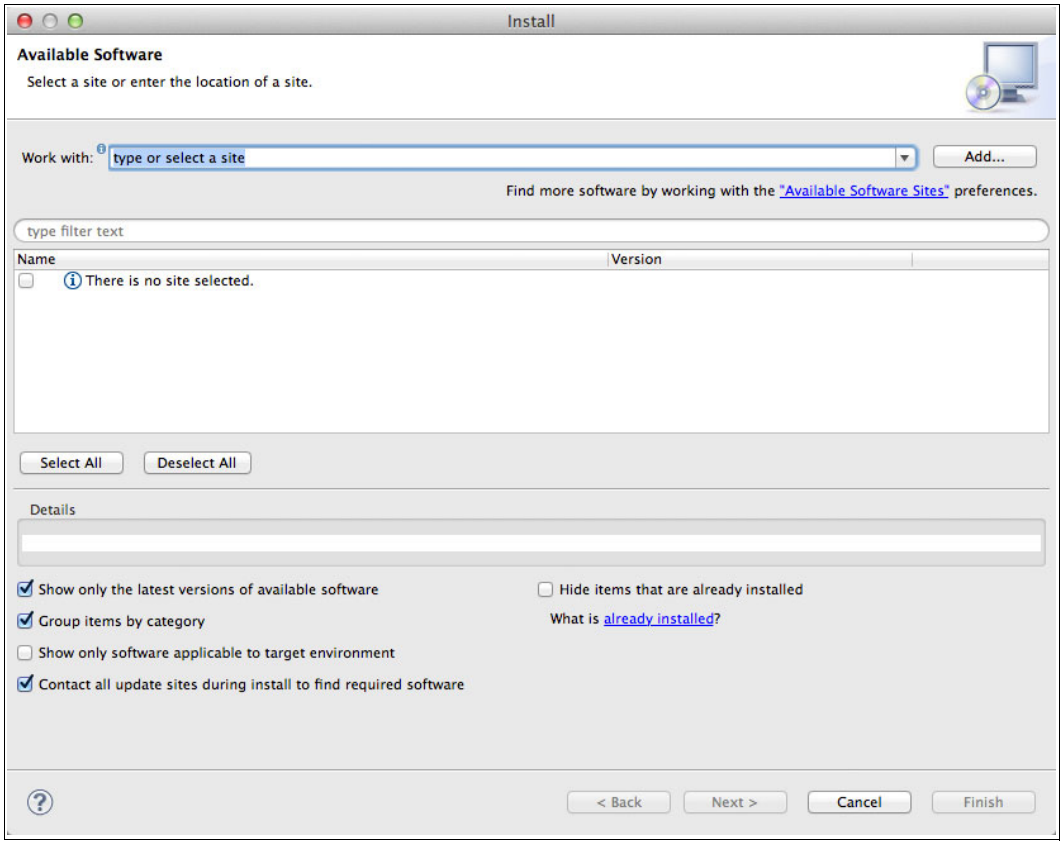

Figure 6-17 Eclipse Available Software window

2. Click **Add** beside the **Work with** list to display the Add Repository window. Enter a repository name, such as `WL Studio 5.0`, and then click **Archive**. This selection indicates that the installation uses an archive file. Locate your IBM Worklight Studio Update Site Archive, select the file, and click **Open**.
3. The name and archive location is now displayed in the Add Repository window. Click **OK** to add this repository to the list of sites that Eclipse uses to locate software to install.
4. The features of IBM Worklight Studio V5.0 are now displayed in the Available Software window. For the purposes of this installation, select all of the features and click **Next**.
5. Review the features to be installed and click **Next.** Review and accept the license agreements and click **Finish** to begin the installation.
6. When the installation is complete, close Eclipse.

IBM Worklight Studio is now installed and ready to use.

6.5 Using the Liberty profile server

In our scenario, both IBM Worklight and WebSphere eXtreme Scale are running with the Liberty profile. The directory `wlp` is the Liberty home directory and the `xs.liberty.tmp`

directory is the WebSphere eXtreme Scale home directory. If you are not familiar with using the Liberty profile server, some of the basics are now described.

If not already defined, you must define the `JAVA_HOME` environment variable to the location of a supported Java Runtime Environment for the Liberty profile. There are two other variable locations that are used in this section:

- *WLP_INSTALL_DIR* - the root directory of the installation
- *WLP_USER_DIR* - the user directory, which contains shared applications, shared configuration files, shared resource definitions, and shared servers.

6.5.1 Creating a Liberty profile server

From the `<WLP_INSTALL_DIR>/bin` directory, run the **`server`** command with the **`create`** parameter and a user-defined server name. For example, to create a Liberty profile server with the name *server1*, run the following command:

```
server create server1
```

The Liberty profile server, *server1*, is now defined under the `<WLP_USER_DIR>/servers` directory.

6.5.2 Configuring a Liberty Profile Server

The `server.xml` file is the main configuration file for the Liberty profile server, as shown in Example 6-18. Use this file to define the features that your applications use. For example, this server, named *server1*, has a single feature that is defined, namely the feature jsp-2.2.

Example 6-18 Example of a simple server.xml

```
<server description="server1">
    <featureManager>
       <feature>jsp-2.2</feature>
    </featureManager>
    <application location="pathtoyourapp/yourwebapp.war"/>
    <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080"
      httpsPort="9443"/>
</server>
```

In addition to the `server.xml` file, you can also use the following configuration files:

- `bootstrap.properties` to define variables such as `listenPort`, which are then referenced in the `server.xml` file.
- `server.env` to define server environment variables, such as `JAVA_HOME`, for the **`server`** command.
- `jvm.options` to define the JVM arguments, such as `Xmx` and `Xms`.

6.5.3 Starting a Liberty profile server

To start the Liberty profile server, issue the following command:

```
server start <servername>
```

where *<servername>* is the name of the Liberty profile server to start.

A

Additional files

The following information is provided:

- Files that are used during the scenario execution that is described in Chapter 3, “Enterprise caching in a mobile environment” on page 23.
- Files that are used during the scenario execution that is described in Chapter 4, “Enterprise caching in an ESB environment” on page 45.
- Source code that is used during the scenario execution to demonstrate the scenario solution that is described in Chapter 5, “Enterprise caching in an extreme transaction processing environment” on page 83.

Additional files for Chapter 3

In the scenario that is detailed in Chapter 3, “Enterprise caching in a mobile environment” on page 23, several Java classes are used to provide the integration between the mobile application and IBM WebSphere eXtreme Scale. In the chapter text, only snippets are provided to explain the key points. The complete files are listed here.

XSCache.java

Example A-1 shows the XSCache.java class.

Example A-1 XSCache.java

```
import java.io.IOException;
import java.util.logging.Logger;

import org.mozilla.javascript.Scriptable;
import com.ibm.json.java.JSONObject;
import com.ibm.websphere.objectgrid.*;
import com.ibm.websphere.objectgrid.plugins.TransactionCallbackException;
import com.worklight.server.bundle.api.*;
import com.worklight.server.integration.api.*;

public class XSCache {
   private static Logger logger = Logger.getLogger("XSCache");
   private Session session = null;
   private String serverName;
   private String gridName;
   private String mapName;
   private String adapterName;
   private String procedureName;
   private boolean isconnected = false;

   //default constructor
   public XSCache(String server, int port, String gridName, String map,
         String adapter, String procedure) {
      this.serverName = server + ":" + port;
      this.gridName = gridName;
      this.mapName = map;
      this.adapterName = adapter;
      this.procedureName = procedure;
   }

   private boolean isConnected(){
      return isconnected;
   }

   private void connect() throws Exception{
      try {
         ObjectGridManager ogMgr =
         ObjectGridManagerFactory.getObjectGridManager();
         ClientClusterContext ccc = ogMgr.connect(serverName, null, null);
         ObjectGrid grid = ogMgr.getObjectGrid(ccc, gridName);
         this.session = grid.getSession();
         this.isconnected = true;
```

```
      logger.info("Connected to Grid");
   } catch (Exception e) {
      logger.info("Failed to connect to the grid.");
      this.isconnected = false;
      throw e;
   }
}

private void disconnect() {
   if (isConnected()) {
      this.session.close();
       logger.info("Succesfully disconnected");
   }
   this.session = null;
   this.isconnected = false;
}

private Scriptable invokeWLAdapter(String key) {
   DataAccessService das =
      WorklightBundles.getInstance().getDataAccessService();
   ProcedureQName pq = new ProcedureQName(this.adapterName,
      this.procedureName);
   InvocationResult invResult = das.invokeProcedure(pq, "['" + key + "']");
   JSONObject jsonObject = invResult.toJSON();
   return JSObjectConverter.jsonToScriptable(jsonObject);
}

public Object getCachedValue(String key) {
   Scriptable result = null;
   //key can't be null
   if (key == null) {
      return null;
   }

   ObjectMap map ;
   Object obj;
   try {
         connect();
         map = session.getMap(this.mapName);
         obj = map.get(key);
         if (obj != null) {
             //return a cached data
            logger.info("Found " + key + " from Grid");
            String data = (String)obj ;
            JSONObject jo = JSONObject.parse(data);
            result = JSObjectConverter.jsonToScriptable(jo);
            logger.info("Get " + key + " from Grid");
            disconnect ();
            return result;
         else {
            //invoke a WL procedure to get data
            DataAccessService das =
                  WorklightBundles.getInstance().getDataAccessService();
            ProcedureQName pq = new ProcedureQName(this.adapterName,
                  this.procedureName);
```

```
                InvocationResult invResult = das.invokeProcedure(pq,
                        "['" + key + "']");
                JSONObject jsonObject = invResult.toJSON();
                result = JSObjectConverter.jsonToScriptable(jsonObject);
                //cache result in XS
                if (jsonObject != null &&
                   jsonObject.get("isSuccessful").toString() == "true")){
                   map.insert(key,jsonObject.serialize());
                   logger.info("Succesfully put " + key + " in Grid");
                }
      } catch (ObjectGridException e) {
            logger.info("ObjectGridException: Error while putting " + key +" in
                        Grid");
            e.printStackTrace();
      } catch (IOException e) {
            logger.info("Can't serialize object. Error while putting value in
Grid");
      } catch (TransactionCallbackException e) {
            return invokeWLAdapter(key);
      } finally {
            disconnect();
      }
      return result;
   }
}
```

Search-impl.js

The `Search-impl.js` file contains two functions:

- trim()
- searchEmployee()

The `trim()` function is a convenience function only to trim leading and trailing blanks. The `searchEmployee()` function calls the cache wrapper class, XSCache, to retrieve the cached value. See Example A-2 for the Search-impl.js file.

Example A-2 Search implementation in Search-impl.js

```
function trim (that) {
      return that.replace(/^\s+|\s+$/g,'');
};

function searchEmployee(searchString) {
      var splitArray,resultArray = new Array();
      var searchResult,success = false;
      var serverName = WL.Server.configuration["cache.server.name"];
      var serverPort = WL.Server.configuration["cache.server.port"];
      var gridName = WL.Server.configuration["cache.grid.name"];
      var mapName = WL.Server.configuration["cache.map.name"];
         var splitArray = searchString.split('OR');
   //insert this for using eXtreamScale
      var XSCache = com.ibm.itso.saw215.XSCache(serverName, serverPort,
            gridName, mapName, "searchEmployee", "search");
   var i;
```

```
   for (i = 0; i < splitArray.length; i++) {
      searchResult = XSCache.getCachedValue(trim(splitArray[i]));
      //original code without eXtreamScale
      /*
       searchResult = WL.Server.invokeProcedure({
         adapter:'searchEmployee',
         procedure:'search',
         parameters:[trim(splitArray[i])]
      });
      */
      if (searchResult !== null) {
         success = searchResult.isSuccessful;
         resultArray = resultArray.concat(searchResult.resultSet);
      }
   }
   //return searchResult;
   return {"isSuccessful":success, "resultSet": resultArray};
}
```

Sample Worklight ant build script

Example A-3 shows the ant script for build and deployment.

Example A-3 Ant script for build and deployment

```
<project name="projectname" basedir="." default="all">
      <target name="compile">
        ...
      </target>
      <target name="build-app" depends="compile">
      <taskdef resource="com/worklight/ant/defaults.properties">
         <classpath>
            <pathelement location="/pathto/worklight-ant.jar" />
         </classpath>
      </taskdef>

      <war-builder projectfolder="." destinationfolder="output"
            warfile="output/${ant.project.name}.war" />
      <customization-builder projectfolder="${ant.project.name}"
            classesfolder="bin/classes"
            destinationfolder="output" />
      <adapter-builder folder="adapters/http" destinationfolder="output"/>
      <app-builder applicationFolder="apps/CoolTest" outputFolder="output"/>


      <app-deployer worklightserverhost=
            "http://${worklightserver.host}:${worklightserver.port}
            /${ant.project.name}"
            deployable="${ant.project.name}.wlapp"></app-deployer>
      <adapter-deployer
            worklightserverhost=
            "http://${worklightserver.host}:${worklightserver.port}
            /${ant.project.name}" deployable="output/http.adapter">
      </adapter-deployer>
```

```
  </target>

  <target name="update-customization">
      <ftp server="${worklightserver.host}"
                  port="${FTP.port}"
            remotedir="${wlp.shared.dir}/lib"
                  userid="${username}"
                  password="${passwd}"
                  passive="yes"
                  depends="yes"
                  binary="yes">
              <fileset dir="output">
                <include name="${customization-file}.jar"/>
              </fileset>
     </ftp>
```

Additional files for Chapter 4

Several files are needed to complete the configuration of the DataPower XI52 Appliance as part of the solution in Chapter 4, “Enterprise caching in an ESB environment” on page 45. Create files with the specific file names and the file contents shown. These files are then uploaded as part of the configuration steps.

CLIENTINFO.wsdl

Example A-4 shows the CLIENTINFO.wsdl file.

Example A-4 CLIENTINFO.wsdl

```
<?xml version="1.0" encoding="UTF-8"?><wsdl:definitions
targetNamespace="http://sw215/clientinformation"
xmlns:bons0="http://instream.ru/mtopup/clientinformation"
xmlns:bons1="http://SW215_CH4" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://sw215/clientinformation"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://sw215/clientinformation"
xmlns:bons0="http://sw215/clientinformation">
            <s:import namespace="http://SW215_CH4"
schemaLocation="ClientInfo.xsd"/>
            <s:element name="GetClientInformation">
        <s:complexType>
          <s:sequence>
            <s:element maxOccurs="1" minOccurs="0" name="ctn" type="s:string"/>
            <s:element maxOccurs="1" minOccurs="0" name="requestID"
type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
```

```
      <s:element name="GetClientInformationResponse">
        <s:complexType>
          <s:sequence>
            <s:element maxOccurs="1" minOccurs="0" name="clientInfo"
type="bons1:ClientInfo"/>
                        <s:element name="subscriptionType" nillable="true"
type="s:int"/>
                        <s:element name="customerTypesId" nillable="true"
type="s:unsignedByte"/>
                    </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://microsoft.com/wsdl/types/">
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetClientInformationSoapIn">
    <wsdl:part element="tns:GetClientInformation" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="GetClientInformationSoapOut">
    <wsdl:part element="tns:GetClientInformationResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="ClientInformationWSSoap">
    <wsdl:operation name="GetClientInformation">
      <wsdl:input message="tns:GetClientInformationSoapIn"/>
            <wsdl:output message="tns:GetClientInformationSoapOut"/>
        </wsdl:operation>
  </wsdl:portType>
    <wsdl:binding name="ClientInformationWSSoap"
type="tns:ClientInformationWSSoap">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetClientInformation">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
    <wsdl:service name="ClientInformationWS">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"/>
    <wsdl:port binding="tns:ClientInformationWSSoap"
name="ClientInformationWSSoap">
      <soap:address location="http://localhost/ClientInformationWS"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

ClientInfo.xsd

Example A-5 shows the ClientInfo.xsd file.

Example A-5 ClientInfo.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://SW215_CH4"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:complexType name="ClientInfo">
      <xsd:sequence>
         <xsd:element minOccurs="0" name="billingId"
            type="xsd:short">
         </xsd:element>

         <xsd:element minOccurs="0" name="marketCode"
            type="xsd:string">
         </xsd:element>
         <xsd:element minOccurs="0" name="regionId"
            type="xsd:short">
         </xsd:element>
         <xsd:element minOccurs="0" name="isRegistered"
            type="xsd:boolean">
         </xsd:element>
         <xsd:element minOccurs="0" name="accountType"
            type="xsd:int">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

ClientInfoBE.wsdl

Example A-6 shows the ClientInfoBE.wsdl file.

Example A-6 ClientInfoBE.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2012 rel. 2 sp1 (http://www.altova.com) by youknow
(volvl) -->
<wsdl:definitions xmlns:bons1="http://SW215_CH4"
xmlns:tns="http://SW215_CH4/ClientInfoBE"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="ClientInfoBE"
targetNamespace="http://SW215_CH4/ClientInfoBE">
   <wsdl:types>
      <xsd:schema targetNamespace="http://SW215_CH4/ClientInfoBE">
         <xsd:import namespace="http://SW215_CH4"
schemaLocation="ClientInfo.xsd"/>
         <xsd:element name="GetClientInformation">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="ctn" nillable="true" type="xsd:string"/>
               </xsd:sequence>
```

```
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="GetClientInformationResponse">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="clientInfo" nillable="true"
type="bons1:ClientInfo"/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="GetClientInformationRequestMsg">
      <wsdl:part name="GetClientInformationParameters"
element="tns:GetClientInformation"/>
   </wsdl:message>
   <wsdl:message name="GetClientInformationResponseMsg">
      <wsdl:part name="GetClientInformationResult"
element="tns:GetClientInformationResponse"/>
   </wsdl:message>
   <wsdl:portType name="ClientInfoBE">
      <wsdl:operation name="GetClientInformation">
         <wsdl:input name="GetClientInformationRequest"
message="tns:GetClientInformationRequestMsg"/>
         <wsdl:output name="GetClientInformationResponse"
message="tns:GetClientInformationResponseMsg"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="ClientInfoBEWSSoap" type="tns:ClientInfoBE">
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="GetClientInformation">
         <soap:operation
soapAction="http://SW215_CH4/ClientInfoBE/GetClientInformation"/>
         <wsdl:input name="GetClientInformationRequest">
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="GetClientInformationResponse">
            <soap:body use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="ClientInfoBEWS">
      <wsdl:port name="ClientInfoBEWSSoap" binding="tns:ClientInfoBEWSSoap">
         <soap:address location="http://localhost/ClientInfoBEWS"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

cache-config.xml

This file defines parameters to access the DataPower XC10, its host name, grid, credentials, and map structure. See Example A-7 for the cache-config.xml file.

Example A-7 cache-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for redistribution
by customer, as part of such an application, in customer's own products.

5724-J34 (C) COPYRIGHT International Business Machines Corp. 2011
All Rights Reserved * Licensed Materials - Property of IBM
-->
<cache url="9.184.118.128" timeout="30" resource="resources/datacaches"
grid="bankacc" username="xcadmin" password="xcadmin" >
    <entry type="getUserBankDetails" ttl="600"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:bank="http://bank/">
      <key label="arg1"
xpath="/soap:Envelope/soap:Body/bank:getUserBankDetails/arg1" />
   </entry>
</cache>
```

cache-get.xsl

See Example A-8 for the cache-get.xsl file.

Example A-8 cache-get.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for redistribution
by customer, as part of such an application, in customer's own products.

5724-J34 (C) COPYRIGHT International Business Machines Corp. 2011
All Rights Reserved * Licensed Materials - Property of IBM
-->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:dyn="http://exslt.org/dynamic"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:bank="http://bank/"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp dyn">

    <xsl:import href="cache-util.xsl" />
```

```
    <xsl:template match="/">

        <xsl:variable name="type"
select="local-name(/soap:Envelope/soap:Body/*[1])" />
        <xsl:variable name="entry-config" select="$cache-config/cache/entry[@type
= $type]" />

        <xsl:message dp:priority="debug">Cache: Message type '<xsl:value-of
select="$type" />' found '<xsl:value-of select="count($entry-config)" />'
configuration item(s)</xsl:message>

        <xsl:choose>
            <xsl:when test="$entry-config">

                <xsl:variable name="key">
                    <xsl:variable name="input" select="/" />
                    <xsl:for-each select="$entry-config/key">
                        <xsl:variable name="xpath" select="@xpath" />
                        <xsl:variable name="key-value">
                            <xsl:for-each select="$input">
                                <xsl:value-of select="dyn:evaluate($xpath)" />
                            </xsl:for-each>
                        </xsl:variable>
                       <xsl:message dp:priority="debug">Cache: Key '<xsl:value-of
select="@label" />' found value '<xsl:value-of select="$key-value"
/>'</xsl:message>
                        <xsl:if test="$key-value">
                            <xsl:if test="dp:local-variable('use-separator')">
                                <xsl:text>;</xsl:text>
                            </xsl:if>
                            <xsl:value-of select="concat(@label,'=',$key-value)"
/>
                            <dp:set-local-variable name="'use-separator'"
value="true()"/>
                        </xsl:if>
                    </xsl:for-each>
                </xsl:variable>
                <xsl:variable name="key-encoded"
select="dp:encode($key,'base-64')" />

               <xsl:message dp:priority="debug">Cache: Created key '<xsl:value-of
select="$key" />'</xsl:message>

                <xsl:variable name="cache-result">
                    <xsl:call-template name="cacheGet">
                        <xsl:with-param name="type" select="$type" />
                        <xsl:with-param name="key" select="$key-encoded" />
                    </xsl:call-template>
                </xsl:variable>

                <xsl:message dp:priority="debug">Cache: Query result:
'<dp:serialize select="$cache-result" omit-xml-decl="yes" />'</xsl:message>

                <xsl:choose>
                    <xsl:when test="normalize-space($cache-result)">
```

```
                        <dp:set-variable name="'var://service/mpgw/skip-backside'"
value="'true'" />

                        <xsl:copy-of select="$cache-result"/>
                    </xsl:when>
                    <xsl:otherwise>
                        <dp:set-variable name="'var://context/cache/type'"
value="$type" />
                        <dp:set-variable name="'var://context/cache/key'"
value="$key-encoded" />
                        <dp:set-variable name="'var://context/cache/ttl'"
value="$entry-config/@ttl" />

                        <xsl:copy-of select="."/>
                    </xsl:otherwise>
                </xsl:choose>

            </xsl:when>
            <xsl:otherwise>

                <xsl:copy-of select="."/>

            </xsl:otherwise>
        </xsl:choose>

    </xsl:template>

</xsl:stylesheet>
```

cache-set.xsl

This file is used in processing logic to save aggregated web-service response. See Example A-9 for the cache-set.xsl file.

Example A-9 cache-set.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for redistribution
by customer, as part of such an application, in customer's own products.

5724-J34 (C) COPYRIGHT International Business Machines Corp. 2011
All Rights Reserved * Licensed Materials - Property of IBM
-->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp">

    <xsl:import href="cache-util.xsl" />
```

```
    <xsl:template match="/">

       <xsl:variable name="type" select="dp:variable('var://context/cache/type')"
/>
        <xsl:variable name="key" select="dp:variable('var://context/cache/key')"
/>
        <xsl:variable name="ttl" select="dp:variable('var://context/cache/ttl')"
/>

        <xsl:if test="$type">
            <xsl:variable name="cache-result">
                <xsl:call-template name="cacheSet">
                    <xsl:with-param name="type" select="$type" />
                    <xsl:with-param name="key" select="$key" />
                    <xsl:with-param name="ttl" select="$ttl" />
                    <xsl:with-param name="data" select="/" />
                </xsl:call-template>
            </xsl:variable>
        </xsl:if>

        <xsl:copy-of select="."/>

    </xsl:template>

</xsl:stylesheet>
```

backend-stuff.xsl

See Example A-10 for the backend-stuff.xsl file.

Example A-10 backend-stuff.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions"
xmlns:func="http://exslt.org/functions"
xmlns:dpfunc="http://www.datapower.com/extensions/functions"
xmlns:dpconfig="http://www.datapower.com/param/config"
extension-element-prefixes="dp func dpfunc dpconfig" exclude-result-prefixes="dp
func dpfunc dpconfig" version="1.0">
   <xsl:template match="/">
      <xsl:variable name="ctn"
select="string(./*[local-name()='Envelope']/*[local-name()='Body']/*[local-name()=
'GetClientInformation']/*[local-name()='ctn']/text())"/>
      <dp:set-variable name="'var://context/CACHE/ctn'" value="$ctn"/>
      <!-- call backend web-service -->
      <xsl:variable name="ws-call">
         <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cli="http://SW215_CH4/ClientInfoBE">
            <soapenv:Header/>
            <soapenv:Body>
               <cli:GetClientInformation>
                  <ctn>
                     <xsl:value-of select="$ctn"/>
```

```
                  </ctn>
               </cli:GetClientInformation>
            </soapenv:Body>
         </soapenv:Envelope>
      </xsl:variable>
      <xsl:variable name="ws-call-result">
         <dp:url-open target="http://172.17.211.197:8088/mockClientInfoBEWSSoap"
response="xml" content-type="text/xml; charset=utf-8">
            <xsl:copy-of select="$ws-call"/>
         </dp:url-open>
      </xsl:variable>
      <dp:set-variable name="'var://context/CACHE/ws-call-result'"
value="$ws-call-result"/>
      <!-- call database -->
      <xsl:variable name="sql-result">
         <dp:sql-execute source="'DB2DS'" statement="'SELECT * FROM
ADMINISTRATOR.CUSTOMER WHERE CTN = ? '">
            <arguments>
               <argument type="SQL_CHAR" mode="INPUT">
                  <xsl:value-of select="$ctn"/>
               </argument>
            </arguments>
         </dp:sql-execute>
      </xsl:variable>
      <dp:set-variable name="'var://context/CACHE/sql-result'"
value="$sql-result"/>
      <xsl:choose>
         <xsl:when test="$sql-result/sql[@result = 'error']">
            <!--- treat error – e.g. write to variable -->
            <dp:set-variable value="$result/sql/message/text()"
name="'var://service/error-message'"/>
         </xsl:when>
         <xsl:otherwise>
            <!-- generate WS response -->
            <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cli="http://sw215/clientinformation">
               <soapenv:Header/>
               <soapenv:Body>
                  <cli:GetClientInformationResponse>
                     <xsl:copy-of
select="$ws-call-result/*[local-name()='Envelope']/*[local-name()='Body']/*[local-
name()='GetClientInformationResponse']/*[local-name()='clientInfo']"/>
                     <cli:subscriptionType>
                        <xsl:value-of
select="$sql-result/sql/row/column[./name/text()
='SUBSCRIPTIONTYPE']/value/text()"/>
                     </cli:subscriptionType>
                     <cli:customerTypesId>
                        <xsl:value-of
select="$sql-result/sql/row/column[./name/text()
='CUSTOMERTYPESID']/value/text()"/>
                     </cli:customerTypesId>
                  </cli:GetClientInformationResponse>
               </soapenv:Body>
```

```
            </soapenv:Envelope>
        </xsl:otherwise>
      </xsl:choose>
   </xsl:template>
</xsl:stylesheet>
```

cache-util.xsl

See Example A-11 for the cache-util.xsl file.

Example A-11 cache-util.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
This sample program is provided AS IS and may be used, executed, copied and
modified without royalty payment by customer (a) for its own instruction and
study, (b) in order to develop applications designed to run with an IBM
WebSphere product, either for customer's own internal use or for redistribution
by customer, as part of such an application, in customer's own products.

5724-J34 (C) COPYRIGHT International Business Machines Corp. 2011
All Rights Reserved * Licensed Materials - Property of IBM
-->
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:str="http://exslt.org/strings"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp str">

    <xsl:variable name="cache-config"
select="document('local:///cache-config.xml')" />
    <xsl:variable name="cache-url" select="$cache-config/cache/@url" />
    <xsl:variable name="cache-timeout" select="$cache-config/cache/@timeout" />
    <xsl:variable name="cache-resource" select="$cache-config/cache/@resource" />
    <xsl:variable name="cache-grid" select="$cache-config/cache/@grid" />
    <xsl:variable name="cache-username" select="$cache-config/cache/@username" />
    <xsl:variable name="cache-password" select="$cache-config/cache/@password" />

    <xsl:template name="cacheGet">
        <xsl:param name="type" />
        <xsl:param name="key" />

        <xsl:call-template name="cacheCall">
            <xsl:with-param name="type" select="$type" />
            <xsl:with-param name="key" select="$key" />
            <xsl:with-param name="method" select="'GET'" />
        </xsl:call-template>
    </xsl:template>

    <xsl:template name="cacheSet">
        <xsl:param name="type" />
        <xsl:param name="key" />
        <xsl:param name="ttl" />
        <xsl:param name="data" />
```

```
        <xsl:call-template name="cacheCall">
            <xsl:with-param name="type" select="$type" />
            <xsl:with-param name="key" select="$key" />
            <xsl:with-param name="method" select="'POST'" />
            <xsl:with-param name="ttl" select="$ttl" />
            <xsl:with-param name="data" select="$data" />
        </xsl:call-template>
    </xsl:template>

    <xsl:template name="cacheCall">
        <xsl:param name="type" />
        <xsl:param name="key" />
        <xsl:param name="method" />
        <xsl:param name="ttl" />
        <xsl:param name="data" />

        <xsl:variable name="parameters">
            <xsl:choose>
                <xsl:when test="number($ttl) > 0">
                    <xsl:value-of select="concat('ttl=',$ttl,'&amp;xml=true')" />
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="'xml=true'" />
                </xsl:otherwise>
            </xsl:choose>
        </xsl:variable>

        <xsl:variable name="url"
select="concat('http://',$cache-url,'/',$cache-resource,'/',$cache-grid,'/',$cache
-grid,'_',$type,'.LUT/',$key,'?',$parameters)" />

        <xsl:variable name="headers">
            <header name="Authorization">Basic <xsl:value-of
select="dp:encode(concat($cache-username,':',$cache-password),'base-64')"
/></header>
            <header name="Content-Type"><xsl:value-of
select="dp:http-request-header('Content-Type')" /></header>
            <xsl:call-template name="loadCookies" />
        </xsl:variable>

        <xsl:variable name="result">
            <dp:url-open response="responsecode" target="{$url}"
http-headers="$headers" timeout="{$cache-timeout}">
                <xsl:copy-of select="$data" />
            </dp:url-open>
        </xsl:variable>

        <xsl:call-template name="saveCookies">
            <xsl:with-param name="cookieHeader"
select="$result/url-open/headers"/>
        </xsl:call-template>

        <xsl:if test="$result/url-open/responsecode = '200'">
```

```
            <dp:set-response-header name="'Content-Type'"
value="$result/url-open/headers/header[@name = 'Content-Type']"/>
            <xsl:copy-of select="$result/url-open/response/*" />
        </xsl:if>

    </xsl:template>

    <!-- - - - - - - - - - - - - - - - - -->

    <xsl:template name="loadCookies">

        <xsl:variable name="cookies"
select="dp:variable('var://system/cache/cookies')"/>

        <xsl:if test="count($cookies/cookies/cookie) > 0">
            <header name="Cookie">
                <xsl:for-each select="$cookies/cookies/cookie">
                    <xsl:value-of select="concat(@name,'=',text(),';')"/>
                </xsl:for-each>
            </header>
        </xsl:if>

    </xsl:template>

    <xsl:template name="saveCookies">
        <xsl:param name="cookieHeader"/>

        <xsl:variable name="cookieList"
select="dp:variable('var://system/cache/cookies')"/>

        <xsl:variable name="newCookieList">
            <cookies>
                <xsl:choose>
                    <xsl:when test="count($cookieList/cookies/cookie) = 0">
                        <xsl:for-each select="$cookieHeader/header[@name =
'Set-Cookie']">
                            <cookie>
                                <xsl:attribute name="name">
                                    <xsl:value-of
select="substring-before(text(),'=')"/>
                                </xsl:attribute>
                                <xsl:value-of
select="substring-after(text(),'=')"/>
                            </cookie>
                        </xsl:for-each>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:for-each select="$cookieList/cookies/cookie">
                            <xsl:variable name="cookieListName"
select="string(@name)" />
                            <cookie>
                                <xsl:attribute name="name">
                                    <xsl:value-of select="$cookieListName"/>
                                </xsl:attribute>
                                <xsl:choose>
```

```
                                    <xsl:when test="$cookieHeader/header[@name =
'Set-Cookie' and substring-before(text(),'=') = $cookieListName]">
                                        <xsl:value-of
select="substring-after($cookieHeader/header[@name = 'Set-Cookie' and
substring-before(text(),'=') = $cookieListName],'=')"/>
                                    </xsl:when>
                                    <xsl:otherwise>
                                        <xsl:value-of select="text()"/>
                                    </xsl:otherwise>
                                </xsl:choose>
                            </cookie>
                        </xsl:for-each>
                    </xsl:otherwise>
                </xsl:choose>
            </cookies>
        </xsl:variable>

        <dp:set-variable name="'var://system/cache/cookies'"
value="$newCookieList"/>
    </xsl:template>

</xsl:stylesheet>
```

Source code for Chapter 5

The source code that is shown in this section is provided *as is* and without any warranties. It represents the implementation of the solution that is proposed in Chapter 5, “Enterprise caching in an extreme transaction processing environment” on page 83.

objectgrid.xml

The `objectgrid.xml` file, shown in Example A-12, is the object grid descriptor file. This file defines grids and backing maps. For information about the parameters in the `objectgrid.xml` file, see 6.2.3, “Understanding WebSphere eXtreme Scale parameters” on page 108.

Example A-12 The object grid descriptor file

```
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="GridOR">
      <backingMap name="OrdersRepository" lockStrategy="PESSIMISTIC"
copyMode="NO_COPY" />
   </objectGrid>
   <objectGrid name="GridOQ">
      <backingMap name="OrdersQueueAF" lockStrategy="PESSIMISTIC"/>
      <backingMap name="OrdersQueueGL" lockStrategy="PESSIMISTIC"/>
      <backingMap name="OrdersQueueMS" lockStrategy="PESSIMISTIC"/>
      <backingMap name="OrdersQueueTZ" lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
```

```
</objectGridConfig>
```

deployment.xml

The `deployment.xml` file, which is shown in Example A-13, defines partitioning of the maps.

Example A-13 The deployment file

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="GridOR">
        <mapSet name="mapSetOR" numberOfPartitions="13" minSyncReplicas="0"
maxSyncReplicas="1" >
         <map ref="OrdersRepository"/>
      </mapSet>
    </objectgridDeployment>
   <objectgridDeployment objectgridName="GridOQ">
        <mapSet name="mapSetOQ" numberOfPartitions="1" minSyncReplicas="0"
maxSyncReplicas="1" >
         <map ref="OrdersQueueAF"/>
         <map ref="OrdersQueueGL"/>
         <map ref="OrdersQueueMS"/>
         <map ref="OrdersQueueTZ"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

OrderRecord

Example A-14 shows the `OrderRecord.java` class, which represents the stock buy or sell purchase order.

Example A-14 OrderRecord class

```
public class OrderRecord implements Externalizable {

   private static final long serialVersionUID = 1L;

   public static int RECEIVED = 1;
   public static int QUEUED = 2;
   public static int PROCESSED = 3;

   //Value fields
   private String orderID;
   private long customerID;
   private String stockID;
   private long dateOfOrder;
   private int numberOfStocks;
   private boolean buy = true;
   private int strikingPrice = -1;
   int state = OrderRecord.RECEIVED;
```

```
   public OrderRecord() {}
   public OrderRecord(String orderID, long customerID, String stockID, long
dateOfOrder, int numberOfStocks, boolean buy, int strikingPrice) {
      this.orderID = orderID;
      this.customerID = customerID;
      this.stockID = stockID;
      this.dateOfOrder = dateOfOrder;
      this.numberOfStocks = numberOfStocks;
      this.buy = buy;
      this.strikingPrice = strikingPrice;
   }
   public OrderRecord(String orderID, long customerID, String stockID, long
dateOfOrder, int numberOfStocks, boolean buy) {
      this.orderID = orderID;
      this.customerID = customerID;
      this.stockID = stockID;
      this.dateOfOrder = dateOfOrder;
      this.numberOfStocks = numberOfStocks;
      this.buy = buy;
   }
   public void readExternal(ObjectInput in) throws IOException,
   ClassNotFoundException {
      orderID = in.readUTF().intern();
      customerID = in.readLong();
      stockID = in.readUTF().intern();
      dateOfOrder = in.readLong();
      numberOfStocks = in.readInt();
      strikingPrice = in.readInt();
      buy = in.readBoolean();
   }
   public void writeExternal(ObjectOutput out) throws IOException {
      out.writeUTF(orderID);
      out.writeLong(customerID);
      out.writeUTF(stockID);
      out.writeLong(dateOfOrder);
      out.writeInt(numberOfStocks);
      out.writeInt(strikingPrice);
      out.writeBoolean(buy);
   }
   public String getOrderID() {
      return orderID;
   }
   public void setOrderID(String orderID) {
      this.orderID = orderID;
   }
   public long getCustomerID() {
      return customerID;
   }
   public void setCustomerID(long customerID) {
      this.customerID = customerID;
   }
   public String getStockID() {
      return stockID;
   }
   public void setStockID(String stockID) {
```

```
      this.stockID = stockID;
   }
   public long getDateOfOrder() {
      return dateOfOrder;
   }
   public void setDateOfOrder(long dateOfOrder) {
      this.dateOfOrder = dateOfOrder;
   }
   public int getNumberOfStocks() {
      return numberOfStocks;
   }
   public void setNumberOfStocks(int numberOfStocks) {
      this.numberOfStocks = numberOfStocks;
   }
   public boolean isBuy() {
      return buy;
   }
   public void setBuy(boolean buy) {
      this.buy = buy;
   }
   public int getStrikingPrice() {
      return strikingPrice;
   }
   public void setStrikingPrice(int strikingPrice) {
      this.strikingPrice = strikingPrice;
   }
   public int getState() {
      return state;
   }
   public void setState(int state) {
      this.state = state;
   }
}
```

TIC.java

A TIC holds the latest price for a specified stock. The `TIC.java` file, which is shown in Example A-15, defines this TIC object.

Example A-15 TIC class

```
public class TIC implements Serializable {

   private String stockID;
   private int stockPrice; //price in cents

   public TIC (String sID, int sPrice){
      stockID = sID;
      stockPrice = sPrice;
   }
   public String getStockID() {
      return stockID;
   }
   public int getStockPrice() {
```

```
      return stockPrice;
   }
}
```

OrderQueueingAgent.java

The `OrderQueueingAgent.java` class, shown in Example A-16, is an implementation of a WebSphere eXtreme Scale agent. This agent pulls new orders from the Order Repository cache and clears the cache.

Example A-16 OrderQueuingAgent class

```
public class OrderQueueingAgent implements MapGridAgent, Serializable {

   private static final long serialVersionUID = 1L;

   private static final String CLASS_NAME = OrderQueueingAgent.class.getName();

    private static final TraceComponent tc = Tr.register(CLASS_NAME,
            Constants.TR_MONITOR_GROUP_NAME,
            Constants.TR_RESOURCE_BUNDLE_NAME);

 // Values used to filter results
   String stockID = null;
   boolean filterOn = false;

   /**
     * This method is called once per key per partition.
     * IT IS NOT SUPPORTED FOR MAPS WITH PLACEMENT STRATEGY = PER_CONTAINER.
     * @param s The Session to use for any work. Transactions are controlled by
the caller.
    * @param map The ObjectMap this agent was invoked for
    * @param key The key of the entry to process.
    * @return The result of the processing.
    */
   public OrderRecord process(Session s, ObjectMap map, Object key){
      String methodName = "process";
      if (tc.isEntryEnabled()) { Tr.entry(tc, CLASS_NAME+":"+methodName, new
Object[] {map.getName()}); }
      try {
         String key1 = (key instanceof String) ? (String) key : null;
         if ( key1 != null) {
            Object valueO = map.get(key1);
            OrderRecord value =  (valueO instanceof OrderRecord) ? (OrderRecord)
valueO : null;
            if(value != null){
               if (tc.isDebugEnabled()) {
                  Tr.exit(tc, CLASS_NAME+":"+methodName,
"orderID="+String.valueOf(key1)+"/stockID="+value.getStockID()+"/numberOfStocks="+
value.getNumberOfStocks()+"/isBuy="+value.isBuy());
               }
               return value;
            }
         }
```

```
      } catch (ObjectGridException e) {
         FFDCFilter.processException(e, CLASS_NAME + ".process","56");
         if (tc.isDebugEnabled()) { e.printStackTrace(System.out); }
      }

      if (tc.isDebugEnabled()) { Tr.exit(tc, CLASS_NAME+":"+methodName, "null");
}
      return null;
   }
   /**
    * This method is called once per partition.<p>
     * Filter the set of entries by setting values for: srcContainer, partition,
tgtContainer
    * @param s The Session to use for any work. Transactions are controlled by the
caller.
    * @param map The ObjectMap this agent was invoked for.
    * @return A map of processed values for keys
    */
   public Map<String, OrderRecord> processAllEntries(Session s, ObjectMap map){
      String methodName = "processAllEntries";
      if (tc.isEntryEnabled()) { Tr.entry(tc, CLASS_NAME+":"+methodName, new
Object[] {map.getName()}); }

      HashMap<String, OrderRecord> l = new HashMap<String, OrderRecord>();
      try {
         MapIndex allIndex = (MapIndex)
map.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
         Iterator keys = allIndex.findAll();
         while (keys.hasNext()) {
            Object key0 = keys.next();
            String key = (key0 instanceof String) ? (String) key0 : null;
            if ( key != null) {
               Object value0 = map.get(key);
               OrderRecord value =  (value0 instanceof OrderRecord) ?
(OrderRecord) value0 : null;
               if(value != null && value.getState() == OrderRecord.RECEIVED){
                  if(filterOn){
                     if(stockID != null && !stockID.equals(value.getStockID())){
                        continue;
                     }
                  }
                  l.put(key, value);
                  map.remove(key);
               }
            }
         }
      } catch (ObjectGridException e) {
         FFDCFilter.processException(e, CLASS_NAME + ".processAllEntries","137");
         if (tc.isDebugEnabled()) { e.printStackTrace(System.out); }
      }
      if (tc.isDebugEnabled()) {
         Set<String> keySet = l.keySet();
         Iterator<String> iter = keySet.iterator();
         while (iter.hasNext()){
            String key = iter.next();
```

```
            OrderRecord store = l.get(key);
            Tr.debug(tc, CLASS_NAME+":"+methodName,
"orderID="+String.valueOf(key)+"/stockID="+store.getStockID()+"/numberOfStocks="+s
tore.getNumberOfStocks()+"/isBuy="+store.isBuy());
         }
      }

      if (tc.isDebugEnabled()) { Tr.exit(tc, CLASS_NAME+":"+methodName, l.size());
}
      return l;
   }
   public void setStockID(String stockID) {
      this.stockID = stockID;
      filterOn = true;
   }
}
```

OrderQueueing.java

The `OrderQueueing.java` class is shown in Example A-17. This class uses the ObjectQueueingAgent to get the new orders from the Order Repository cache and organizes those orders into queues and stores them in the Order Queues cache.

Example A-17 OrderQueueing class

```
public class OrderQueueing {

   public static HashMap<String, ArrayList> queues;

   public static void initializeQueues() {
      queues = new HashMap<String, ArrayList>();
      String[] stockIDs = MyCompConstants.stockIDs;
      for (int i = 0; i< stockIDs.length; i++){
         if(queues.get(stockIDs[i])==null){
            queues.put(stockIDs[i], new ArrayList<OrderRecord>());
         }
      }
   }
   public static void uploadQueues(Session sess) throws Exception {
      Iterator<String> iter = queues.keySet().iterator();
      while(iter.hasNext()){
         String queueName = (String)iter.next();
         String queueMapName = MyCompConstants.getQueueMapName(queueName);
         ObjectMap map = sess.getMap(queueMapName);
         Object q0 = map.get(queueName);
         if(q0 == null){
            map.insert(queueName, (ArrayList)queues.get(queueName));
         } else {
            ((ArrayList)q0).addAll((ArrayList)queues.get(queueName));
            map.remove(queueName);
            map.insert(queueName, (ArrayList)q0);
         }
         queues.put(queueName, new ArrayList());
      }
   }
```

```
   public static void addOrder2LocalQueue(OrderRecord value){
      Object queueO = queues.get(value.getStockID());
      if(queueO != null){
         ArrayList<OrderRecord> queue = queueO instanceof ArrayList ?
(ArrayList)queueO : null;
         if(queue != null){
            queue.add(value);
         }
      }
   }
   /**
    * @param args
    */
   public static void main(String[] args) {

      Session sess = null, sessQ = null;
      try{
         ClientClusterContext ccc =
ObjectGridManagerFactory.getObjectGridManager().connect(MyCompConstants.XS_CATALOG
_HOST+":2809", null, null);
         ObjectGrid grid =
ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc,
MyCompConstants.GRID_ORDER_REPOSITORY);
         sess = grid.getSession();
         ObjectGrid grid1 =
ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc,
MyCompConstants.GRID_ORDER_QUEUES);
         sessQ = grid1.getSession();
         initializeQueues();
         ObjectMap map = sess.getMap(MyCompConstants.SIDE_CASH_MAP_NAME);
         long start = System.currentTimeMillis();
         long end = start;
         int counter = 0;
         if(map != null) {
            System.out.println("Starting qeueuing Orders!!!!!!!");
            while(true) {
               Map m = map.getAgentManager().callMapAgent(new
OrderQueueingAgent());
               int setSize = m.keySet().size();
               if(setSize > 0){ //do anything only if the map is not empty
                  for (Object key: m.keySet()) {
                     if ( key != null) {
                        Object valueO = m.get(key);
                        OrderRecord  value = (valueO instanceof OrderRecord) ?
(OrderRecord) valueO : null;
                        if(value != null){ //insert order into queue and remove
from the side cache
                           value.setState(OrderRecord.QUEUED);
                           addOrder2LocalQueue(value);
                        }
                     }
                  }
                  uploadQueues(sessQ);
                  counter += setSize;
                  start = System.currentTimeMillis();
```

```
                if(start - end > 10000){
                   System.out.println("Queued "+counter+" Orders in
"+(start-end) +" miliseconds.");
                   end=start;
                   counter = 0;
                }
             }
          }
       }
    } catch (Exception e) {
       e.printStackTrace();
    } finally {
       if (sess != null){
          sess.close();
       }
       if (sessQ != null){
          sessQ.close();
       }
    }
 }
}
```

OrderProcessingAgent.java

The `OrderProcessingAgent.java` class, shown in Example A-18, updates all orders for a specified stock in the Order Queues cache with the latest Striking Price and new state.

Example A-18 OrderProcessingAgent class

```
public class OrderProcessingAgent  implements ReduceGridAgent, Serializable {

private static final long serialVersionUID = 1L;

   private static final String CLASS_NAME = OrderProcessingAgent.class.getName();

    private static final TraceComponent tc = Tr.register(CLASS_NAME,
            Constants.TR_MONITOR_GROUP_NAME,
            Constants.TR_RESOURCE_BUNDLE_NAME);

 // Values used to filter results
   String stockID = null;
   int strikingPrice = 0;
   boolean filterOn = false;

   /**
    * The method should reduce the entries referenced by the key collection
    * to a single value. An example would be a method that returns
    * an average value for an attribute in a collection of objects. Transactions
are managed by
    * the caller. Any exception results in a rollback. If this map is associated
with an entity, then a collection
    * of populated entities are provided to the collection instead Tuple keys. The
class used for the entity
    * is identified using the EntityAgentMixin interface.
```

```
     * @param s The Session to use for any work. Transactions are controlled by
the caller.
    * @param map The ObjectMap this agent was invoked for.
    * @param keys The collection of keys or entities whose values should be
reduced.
    * @return The reduced value calculated from processing the entries specified.
    */
   public Object reduce(Session s, ObjectMap map, Collection/*<Object key>*/
keys){
      return reduce(s, map);
   }
   /**
    * This method is called on each partition. The Agent may have extra state
including
    * possible a query to eliminate some partition entries.
     * @param s The Session to use for any work. Transactions are controlled by
the caller.
    * @param map The ObjectMap this agent was invoked for
    * @return The single value calculated from processing the entries for the
partition.
    */
   public Object reduce(Session s, ObjectMap map){
      String methodName = "reduce";
      if (tc.isEntryEnabled()) { Tr.entry(tc, CLASS_NAME+":"+methodName, new
Object[] {map.getName()}); }
      int numberOrdersProcessed = 0;
      try {
         MapIndex allIndex = (MapIndex)
map.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
         Iterator keys = allIndex.findAll();
         while (keys.hasNext()) {
            Object key0 = keys.next();
            String key = (key0 instanceof String) ? (String) key0 : null;
            if ( key != null && key.equals(stockID)) {
               Object value0 = map.get(key);
               List list =  (value0 instanceof List) ? (List) value0 : null;
               if(list != null && list.size() > 0){
                  for (int i=0; i<list.size();i++){
                     Object order0 = list.get(i);
                     if(order0 != null){
                        OrderRecord order = order0 instanceof OrderRecord ?
(OrderRecord)order0 : null;
                        if(order != null){
                           order.setStrikingPrice(strikingPrice);
                           order.setState(OrderRecord.PROCESSED);
                        }
                     }
                  }
                  map.update(key0, list);
               }
            }
         }
      } catch (ObjectGridException e) {
         FFDCFilter.processException(e, CLASS_NAME + ".processAllEntries","137");
         if (tc.isDebugEnabled()) { e.printStackTrace(System.out); }
```

```
      }
      if (tc.isDebugEnabled()) {
         Tr.debug(tc, CLASS_NAME+":"+methodName,
"stockID="+stockID+"/numberOrdersProcessed="+numberOrdersProcessed);
      }

      if (tc.isDebugEnabled()) { Tr.exit(tc, CLASS_NAME+":"+methodName,
numberOrdersProcessed);  }
      return numberOrdersProcessed;
   }
   /**
    * This is called when further reduction is needed.
    * @param results the intermediate results to reduce.
    * @return The single value calculated from processing the intermediate
results.
    */
   public Object reduceResults(Collection results){
        return results;
   }
   public void setStockID(String stockID) {
      this.stockID = stockID;
      filterOn = true;
   }
   public void setStrikingPrice(int price){
      strikingPrice = price;
   }
}
```

OrderProcessing.java

The `OrderProcessing.java` class is shown in Example A-19. The OrderProcessing class pulls the tics from WebSphere MQ and uses the OrderProcessingAgent to set the striking price on all orders for the corresponding stock. Later, it puts the updated orders on the message queue to be picked up by the Order Execution unit.

Example A-19 OrderProcessing class

```
public class OrderProcessing {


   public static Time sessionClosingTime = Time.valueOf("16:00:00");

   public static void closeSession(HashMap<String, Integer> latestPrices,
ObjectGrid grid) throws Exception {
      for (String key: latestPrices.keySet()){
         Integer price = latestPrices.get(key);
         setStrikingPrice(key, price, grid);
      }
   }
   public static int setStrikingPrice(String stockID, int price, ObjectGrid grid)
throws Exception {
      Session sess = grid.getSession();
      ObjectMap map = sess.getMap(MyCompConstants.getQueueMapName(stockID));
      OrderProcessingAgent agent = new OrderProcessingAgent();
```

```
      agent.setStockID(stockID);
      agent.setStrikingPrice(price);
      int numberOrders = (Integer)map.getAgentManager().callReduceAgent(agent);
      Object orderQueueO = map.get(stockID);
      map.remove(stockID);
      if(orderQueueO != null){
         List<OrderRecord> orderQueue = orderQueueO instanceof List ?
(List)orderQueueO : null;
         if(orderQueue != null){
            sendOffRecords((OrderRecord[])orderQueue.toArray());
         }
      }
      sess.close();
      return numberOrders;
   }
   public static void sendOffRecords(OrderRecord[] records){
      MQQueueManager queueManager = null;
      try {
         MQEnvironment.hostname = MyCompConstants.MQ_MANAGER_HOST;
          MQEnvironment.channel = MyCompConstants.ORDERS_MQ_CHANNEL;
         queueManager=new MQQueueManager(MyCompConstants.MQ_MANAGER);
         int openOptions = MQConstants.MQOO_INPUT_AS_Q_DEF |
MQConstants.MQOO_OUTPUT;
         MQQueue queue =
queueManager.accessQueue(MyCompConstants.ORDERS_QUEUE_NAME, openOptions);

         MQPutMessageOptions pmo = new MQPutMessageOptions();
         for(int i=0; i< records.length; i++){
            MQMessage msg = new MQMessage();
            msg.writeObject(records[i]);
            queue.put(msg, pmo);
            queue.close();
         }
         queueManager.disconnect();
      } catch (Exception e){
         e.printStackTrace();
      }
   }
   /**
    * @param args
    */
   public static void main(String[] args) {

      int threadPoolSize = 50;
      if(args.length>0 && args[0]!=null){
         threadPoolSize = Integer.valueOf(args[0]).intValue();
      }
      ThreadPool threadPool =
ThreadPoolManagerFactory.getThreadPoolManager().getThreadPool("Order processing
thread", 2, threadPoolSize);
      HashMap<String, Integer> latestPrices = new HashMap<String, Integer>();

      MQQueueManager queueManager = null;
      try {
```

```
        ClientClusterContext ccc =
ObjectGridManagerFactory.getObjectGridManager().connect(MyCompConstants.XS_CATALOG
_HOST+":2809", null, null);
        ObjectGrid grid =
ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc,
MyCompConstants.GRID_ORDER_QUEUES);

        MQEnvironment.hostname = MyCompConstants.MQ_MANAGER_HOST;
         MQEnvironment.channel = MyCompConstants.TICS_MQ_CHANNEL;
         MQException.log=null;
        queueManager=new MQQueueManager(MyCompConstants.MQ_MANAGER);
        int openOptions = MQConstants.MQOO_INPUT_AS_Q_DEF |
MQConstants.MQOO_OUTPUT;
        MQQueue queue = queueManager.accessQueue(MyCompConstants.TICS_QUEUE_NAME,
openOptions);
        MQMessage receiveMessage = new MQMessage();
        MQGetMessageOptions getMessageOptions = new MQGetMessageOptions();

        OrderProcessingRunnable orderProcessingRunnable;

        long start = System.currentTimeMillis();
        long end = start;
        DateFormat df = new SimpleDateFormat("HH:mm:ss");
        Date time = new Date(start);
        System.out.println("Starting processing Orders!!!!!!!");
        while (df.format(time).compareTo(sessionClosingTime.toString())<0){
           try {
              queue.get(receiveMessage, getMessageOptions);
              Object TICO = receiveMessage.readObject();
              if(TICO != null){
                 TIC tic = TICO instanceof TIC ? (TIC)TICO : null;
                 if(tic != null){
                    String stockID = tic.getStockID();
                    latestPrices.put(stockID, tic.getStockPrice());
                    orderProcessingRunnable = new OrderProcessingRunnable(tic,
grid);
                    try {
                       threadPool.execute(orderProcessingRunnable);
                    } catch ( Exception e ) {
                       e.printStackTrace();
                    }
                 }
                 end = System.currentTimeMillis();
                 receiveMessage = new MQMessage();
              }
           } catch (MQException e){
              e.printStackTrace();
           }
           time = new Date(end);
        }
        closeSession(latestPrices, grid);
        // Close the queue
        System.out.println("Closing the queue");
        queue.close();
```

```
      } catch (Exception e) {
         e.printStackTrace();
      } finally {

      }
   }
}
```

OrderProcessingRunnable.java

The `OrderProcessingRunnable.java` class represents a runnable, which runs the OrderProcessingAgent. This agent updates new orders with the striking price and new state. This class is shown in Example A-20.

Example A-20 OrderProcessingRunnable class

```
public class OrderProcessingRunnable implements Runnable {

   private TIC tic = null;
   ObjectGrid grid;

   public OrderProcessingRunnable(TIC t, ObjectGrid g){
      tic = t;
      grid = g;
   }
   public void run() {
      try{
         long start = System.currentTimeMillis();
         int ordersNumber = OrderProcessing.setStrikingPrice(tic.getStockID(),
tic.getStockPrice(), grid);
         System.out.println("Send to MQ "+ordersNumber+" Orders in
"+(System.currentTimeMillis()-start) +" miliseconds.");
      } catch (Exception e){
         e.printStackTrace();
      } finally {
      }
   }
   public void stop() {
   }
}
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *User's Guide to WebSphere eXtreme Scale,* SG24-7683
- *IBM WebSphere eXtreme Scale V7: Solutions Architecture,* REDP-4602
- *Elastic Dynamic Caching with the IBM WebSphere DataPower XC10 Appliance,* REDP-4851.

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- WebSphere eXtreme Scale

  http://www-01.ibm.com/software/webservers/appserv/extremescale/
- WebSphere DataPower XC10 Appliance

  http://www-01.ibm.com/software/webservers/appserv/xc10/
- IBM WebSphere eXtreme Scale V8.5 Information Center

  http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r5/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.doc%2Fwelcome%2Fwelcome_xs.html
- WebSphere eXtreme Scale wiki documentation

  https://www.ibm.com/developerworks/wikis/display/objectgrid/Getting+started
- IBM Software technical white paper: Elastic caching for scalability, dynamic growth, and performance

  http://public.dhe.ibm.com/common/ssi/ecm/en/wsw14170usen/WSW14170USEN.PDF
- IBM WebSphere Distributed Caching Products

  http://public.dhe.ibm.com/common/ssi/ecm/en/wsd14110usen/WSD14110USEN.PDF
- Analyst report: Total Economic Impact Of IBM WebSphere eXtreme Scale

  ftp://public.dhe.ibm.com/software/webservers/appserv/extremescale/Forrester-WebSphere-eXtreme-Scale.pdf

- IBM Installation Manager Version 1 Release 5 information center

  http://pic.dhe.ibm.com/infocenter/install/v1r5/index.jsp
- IBM Worklight getting started

  https://www.ibm.com/developerworks/mobile/worklight/getting-started/index.html
- Eight steps to IBM Worklight mobile application development

  http://www.ibm.com/developerworks/websphere/techjournal/1210_chen/1210_chen.html
- WebSphere Education

  http://www.ibm.com/websphere/education
- IBM Education: Developing Applications for IBM WebSphere eXtreme Scale V7.1

  http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=WA974
- IBM Education: WebSphere Expert Arena for IBM WebSphere eXtreme Scale V7

  http://www-304.ibm.com/jct03001c/services/learning/ites.wss/us/en?pageType=course_description&courseCode=WX972
- IBM Education Assistant

  http://www-01.ibm.com/software/info/education/assistant/index.shtml

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Enterprise Caching Solutions using IBM WebSphere DataPower SOA Appliances and IBM WebSphere eXtreme Scale

Introduction to caching

Concepts and considerations

Real-world scenarios

In the dynamic business environment of today, Information Technology (IT) organizations face challenges in addressing around scalability and performance. This IBM Redbooks publication is targeted for IT architects, IT personnel, and developers who are looking to integrate caching technologies, specifically *elastic caching*, into their business environment to enhance scalability and performance. Although it is helpful to know caching technologies, an introduction to caching technologies in general is included. In addition, technical details are provided about implementing caching by using several IBM products.

The IBM WebSphere eXtreme Scale product provides several functions to enhance application performance and scalability. It provides distributed object caching functionality, which is essential for elastic scalability and next-generation cloud environments. It helps applications process massive volumes of transactions with extreme efficiency and linear scalability. By using the scalable in-memory data grid, enterprises can benefit from a powerful, high-performance elastic cache.

The IBM WebSphere DataPower XC10 Appliance enables your business-critical applications to scale cost effectively with consistent performance by using elastic caching in a purpose-built, easy-to-use appliance.

This publication explains the benefits of using various caching techniques in your enterprise, specifically involving the use of IBM WebSphere eXtreme Scale and the IBM WebSphere DataPower XC10 Appliance. Three real-world scenarios are described that use these enterprise caching technologies to solve issues that face the businesses of today.

SG24-8043-00 ISBN 0738438278

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks