Sybase Unwired Platform
Architecture
# TABLE OF CONTENTS

1  SYBASE UNWIRED PLATFORM VISION AND GOALS

1  MOBILE APPLICATION PATTERNS AND PROGRAMMING
   1  Offline Data Pattern
   1  Data Virtualization Pattern
   2  Data Publication Pattern
   2  Operation Replay Pattern
   2  Object-Oriented Application Programming

3  HIGH-LEVEL ARCHITECTURE
   4  Development Paradigm
   4  Mobile Application Stack
   5  Logical Architecture Block Diagram
   5  Server Architecture
   7  Administration Architecture Block Diagram
   7  Sybase Unwired Platform, Firewall, and DMZ
   8  Cache in CDB

10 PERFORMANCE AND DATA MOBILIZATION PATTERNS
   10  Data Mobilization Patterns
   11  Considerations during Design/Development
1 SYBASE UNWIRED PLATFORM VISION AND GOALS
The vision for Sybase Unwired Platform is an integrated mobile middleware that serves the Mobile Enterprise Application Platform market and includes:

1. Support for multiple client devices and mobile operating systems.
2. Support for native clients – device-dependent, but rich in capabilities including an attractive and functional user interface.
3. Support for development of applications on heterogeneous devices.
5. An extensible data mobilization architecture that uses standard and proprietary interfaces to support a variety of enterprise data resources.
6. Unified platform administration
7. End-to-end security, offering protection that extends from the enterprise to devices.
8. Support for mobile work environments that include both users who are occasionally connected and those who are mostly connected (occasionally disconnected).
9. Push notification that alerts clients to synchronize.

2 MOBILE APPLICATION PATTERNS AND PROGRAMMING
Based on extensive mobility experience helping customers implement mobile solution, Sybase has identified the patterns that are important to the development and successful operation of mobile applications. Sybase Unwired Platform, both middleware and tooling, is designed to provide first-class support for these patterns.

2.1 OFFLINE DATA PATTERN
This pattern is fundamental to the usability of a mobile application, regardless of connectivity. It satisfies the requirement that mobile applications can function without wireless connectivity. This pattern has two requirements:

1. Appropriate data must be available on the device
2. Changes or operations against the on-device data persist until the user is done with his or her work and connectivity is available

Even though wireless connectivity is improving, the two main obstacles—battery limitations and high latency—continue to make this pattern a requirement by most mobile applications.

2.2 DATA VIRTUALIZATION PATTERN
The need for mobile applications being able to access back-end data from more than just the enterprise database is growing. Mobile applications may also need to access multiple data resources at the same time. The data virtualization pattern allows mobile application a holistic and uniform view of back-end data resources by:

1. Lowering the barrier to a variety of back-end data resources
2. Supporting the building of mobile applications that can simultaneously interact with multiple enterprise back-end entities.
3. Allowing mobile middleware to:
   a. Cache to reduce back-end load
   b. Tune caches match the characteristics of back-end data sources to maximize data freshness
4. Normalizing data to a relational view
5. Developing composite data model involving multiple back-end data sources
6. Performing differential synchronization
7. Notifying changes to high-valued data subscribed to by user

2.3 DATA PUBLICATION PATTERN

Mobile devices have limited resources and can store only a subset of enterprise data. To load only data that the mobile applications require from the data source, this pattern calls for the use of publications that define the required data subsets. The data virtualization pattern addresses how to make enterprise data available for mobility, whereas the data publication pattern addresses how to select the right data.

2.4 OPERATION REPLAY PATTERN

Because mobile applications must work with a variety of back-end data sources, changes are likely to be performed via service calls (Web Services), specific API invocations, the Java Connector Architecture Common Client Interface, and so on. The operation replay pattern supports either a service-oriented architecture or ad hoc application integration techniques. The mobile application executes an operation on the client side; the mobile infrastructure relays the operation back to the enterprise and executes the server portion on behalf of the mobile application in a robust fashion.

2.5 OBJECT-ORIENTED APPLICATION PROGRAMMING

On the device, data persists as relational tables for mobile applications. These relational tables can be accessed as persistence-capable objects, taking into account the relationships among them. As a result, mobile application developers can manipulate the data on device using Object-Oriented Programming and an object persistence API. The Object API provides more than an abstraction for the relational data access and update; its integrated data synchronization capability may simplify mobile development.

The following diagram shows how Sybase Unwired Platform supports the various mobilization patterns using mobile business objects (MBOs). Development starts with the definition of the MBO and the subset of data that it transparently maintains with the back-end Enterprise Information Systems.

![SUP Mobile Development Model Diagram](image-url)
3 HIGH-LEVEL ARCHITECTURE

Sybase Unwired Platform architecture is based on the principle of factorization, in other words, sharing a common infrastructure created from the “best-of-breed” Sybase mobile technology. The architectural design divides the platform into three separate concerns that constitute the operating environment of a mobile application: Device, Mobile Middleware and Data Services. Platform-wide services—security, administration and development tooling—surround and support these concerns.

Figure 2 - SUP High-Level Architecture

Figure 3 illustrates the functionalities and collaboration points for each concern. Mobile Middleware Services and Device Services interact through relational database synchronization. The use of MBO and Object API abstracts the low-level synchronization details from the mobile application. On the server side, the cache, which is also called the consolidated database, is the interaction point between Data Services and Mobile Middleware Services that supports data mobilization.

Figure 3 - SUP Platform Architecture
3.1 DEVELOPMENT PARADIGM
MBOs are the focal point of mobile application development. They abstract the mobile application developer from interaction with EIS. Eclipse or Visual Studio development tooling assists developers in creating the SUP data model. The data model consists of packages of related MBOs. Each MBO is defined to retrieve a set of data (may be parameterized) from the EIS for use by the mobile application. The tooling supports visual development of device specific mobile applications based on the MBO package with Create, Read, Update and Delete (CRUD) capabilities. For custom development, Mobile Application developers generate device platform-specific code from the MBO definitions. It is through the generated MBO that the application interacts with the data on the device and, in turn, those on the enterprise. The data persistence module supports the generated code.

3.2 MOBILE APPLICATION STACK
A mobile application rests on multiple layers of services that shield the developer from the complexity of running on a mobile device in an occasionally connected environment. The generated classes make available an object relational mapping over the data in the mobile database. The integrated stack provides critical mobile services required by mobile applications and device management. It is designed to work with the SUP runtime infrastructure.

Figure 4 illustrates the client-side stack architecture. The Afaria client runs as a separate process and provides security and device management services on the device. Synchronization with Sybase Unwired Platform is provided by the UltraLite® database (ULDB).
3.3 LOGICAL ARCHITECTURE BLOCK DIAGRAM

Figure 5 shows the different blocks within the three concerns of the Sybase Unwired Platform architecture. The administration component and the Eclipse and Visual Studio development tooling are not included in the illustration. Mobile Middleware Services and Data Services are designed to run within a single process as the Unwired Server. Scalability is achieved by running multiple Unwired Servers behind a load balancer.

The consolidated database (CDB) is used by Data Services to store the metadata and mobilized data that supports the data virtualization pattern. Data from back-end resources is mobilized as MBOs. These objects are created by extracting data from the resources and caching it in the CDB as tables. By default, the cache maintained by Data Services uses the write-invalidate strategy.

Data Services connects to back-end data resources using these interfaces:
- Enterprise databases – JDBC
- Services within Service Oriented Architecture – Web Services
- Enterprise Information Systems – Java Connector Architecture

Multiple Data Services share a single CDB. To protect against single point of failure, Sybase recommends that you run the CDB within a Microsoft cluster. Alternately, customers can choose to deploy SQL Anywhere® in mirrored high availability (hot standby) mode. From an architectural point of view, the platform has no single point of failure.

3.4 SERVER ARCHITECTURE

Unwired Server
Database Synchronization

Data Synchronization Services is the component within Mobile Middleware Services that provide relational database synchronization between a device’s UltraLite® database and the CDB. During start-up, the Unwired Server process starts the JVM, which hosts Mobile Middleware Services and Data Services. All Unwired Server processes within a Sybase Unwired Platform cluster connect to the same CDB.
Web Container
The Web container handles data change notifications from EIS. It is also used for administration purpose.

Mobile Middleware Services
Mobile Middleware Services provide the handlers that are invoked during the upload and download phases of synchronization. The handlers, in turn, interact with the EIS through Data Services. During the upload phase, Mobile Middleware Services perform the operation replay by invoking appropriate MBO Create/Update/Delete operations. During the download phase, it interacts with Data Services to determine the set of data to be downloaded to the device’s UltraLite® database.

Data Services
Data Services mediates between Mobile Middleware Services and all varieties of back-end data resources. It supports the data virtualization pattern and offers a relational view of data from the back end. It does using a cache within the CDB. On a per-MBO basis, the cache can be configured to perform write-invalidate or write-through operations. Data Services, based on the cache configuration, determines when to retrieve data from the Enterprise Information Systems, by performing a differential synchronization calculation.

Data Services can apply an attribute or result-set filter to the data retrieved from back-end data resources. The filtered data then fills the cache. The attribute filter uses the exposed filter API and can be developed by customers or System Integrators.

Afaria Server
Afaria Server, which is not part of the Unwired Server, handles device management. Administration for Afaria is via a Web console that can be launched from SUP administration console. See the Afaria product documentation for more information regarding its functionality and administration.

Figure 6 – Server Architecture
3.5 ADMINISTRATION ARCHITECTURE BLOCK DIAGRAM

Figure 7 shows the Sybase Unwired Platform Administration using a Unified Agent Framework (UAF) agent. Sybase Control Center is a management agent that provides administration for multiple Sybase products, including the Sybase Unwired Platform. The UAF agent, which is based on JMX agent technology, requires MBeans to plug into the instrumentation level to handle resource management. Sybase Unwired Platform includes the MBean server.

![Figure 7 – Sybase Unwired Platform Administration Architecture](image)

3.6 SYBASE UNWired PLATFORM, FIREWALL, AND DMZ

Sybase customers have long been concerned about the number of ports that need to be opened in the firewall to support mobile applications. Since there are multiple servers within Sybase Unwired Platform, we must address this concern. Sybase Unwired Platform resolves through the use of Relay Server. Mobile devices supported by Sybase Unwired Platform connect to Relay Server only in the DMZ using HTTPS. Typically, a hardware load balancer spreads the load across a farm of Relay Servers. Each Relay Server routes requests to individual servers within the SUP server farms. Each server type, for example, Unwired Server, Afaria, and so on, has its own server farm. The RSOE, relay server outbound enabler makes an outbound connection to the Relay Server in the DMZ, thus avoiding opening a port in the internal firewall.

Note: Sybase customers are not required to run Relay Server, and may instead use their own load balancers or reverse proxies.

Server failure within a server farm requires clients to reconnect and perform the work again. What the client needs to do in the event of a failure depends on the server it is connecting to.
Virtual Data Model

Data virtualization introduces a layer that represents various back-end data resources as relational tables to:

- Normalize all structured data sources to relational to reduce complexity for tooling and developers
- Allow relational data synchronization to be used as the method of exchanging data between the enterprise and devices

You can develop a mobile application by first identifying the back-end data resources that it needs to leverage. Create an MBO for each data subset that is backed by a data resource. Related MBOs are associated to each other through the definition of appropriate mappings. From the set of MBOs, the Sybase Unwired Platform tooling creates a virtual relational view. However, this view or layer is not exposed to the developers unless customization or special configuration is required.

Cache

Through the tooling, the virtual view developed is deployed into the platform. This view resides in the CDB. During deployment, metadata that represents the package of MBOs is created and persists in the CDB. Tables to store the data retrieved during synchronization are also created during deployment. Developers can control the operation of the cache by specifying:

- Valid duration – time period during which cached data is considered to be current.
- Refresh schedule – whether the cache is automatically refreshed from the back end automatically. If there is no schedule, the cache is refreshed on demand

During synchronization, data from the Enterprise Information Systems is always written to the cache so that any changes since the retrieval can be determined. This methodology supports differential synchronization, in other words, sending to the devices only the data that has changed. If the valid duration is set to zero, data is retrieved from the
Enterprise Information Systems and differences are calculated every time the user synchronizes. Although this pattern may place a substantial load on the Enterprise Information Systems, it always returns the latest information to the user. Developers should understand the application data usage and configure the cache appropriately.

**Bulk-Load Cache**

To match the data mobilization pattern and allow data to efficiently flow from the Enterprise Information Systems to devices, there are two types of cache: bulk-load and partitioned. A bulk-load cache facilitates:

- Efficient bulk retrieval of data from the EIS, which avoids impacting Enterprise Information Systems performance through frequent retrieval of data.
- Sharing of data between multiple mobile users and applications. Typically, the mobile application filters the data it needs through parameters passed to the SUP servers during synchronization.

Bulk-load caches use a filling and filtering pattern. The bulk retrieval fills the cache, and users filter and only download the information they need onto the device. The bulk-load cache assumes that data retrieved from the EIS remains valid for a reasonable duration, so that the bulk-retrieved data can be shared over that amount of time.

**Partition Cache**

When data is partitioned among mobile users, in other words, each user has his or her own set of data, the partition cache feature is used. For example, every field engineer has a unique set of tasks assigned. There is no sharing between users. Furthermore, when the field engineer completes one task and synchronizes to download the next assignment, the platform has to access the Enterprise Information Systems at that instance. If a valid duration is set for the cache, the mobile user cannot retrieve the latest task. It is highly disruptive to users if the cache cannot be split into multiple partitions. While a bulk-load operation can retrieve data for multiple users at a time, the cost of processing the extra data is wasted. It is far more efficient for users to get partitioned data (a limited amount) when he or she synchronizes.

For personalized download data (not shared between users), it is most efficient to use a partition cache. The Enterprise Information Systems must support such an interaction pattern. Incidentally, we have come across EIS that are supporting partition data does not offer any means to retrieve information on a bulk basis (across users). This is another reason why partition cache is developed to match this particular data mobilization pattern.

**Mixed-Mode Caching**

MBOs are contained within a package not only for deployment, but also to make sure that data that corresponds to a certain function scope is co-located. Therefore, SUP allows MBOs within a package to utilize different cache modes. Some MBOs can be employed using bulk load, while others use partitioning. It is the responsibility of the developer to determine which mode is appropriate on a MBO by MBO basis.

**Write-Through Cache**

By default, the cache operates in write-invalidate mode when it processes operation replay from the device. This means that the cache associated with the MBO is invalidated whenever there is an operation replay (Create/Update/Delete). This raises the maintenance cost of the cache, especially if operation replay is very frequent. To alleviate the invalidation and subsequent EIS retrieval, Sybase suggests that developers use write-through mode. However, write-through mode is not an approximation of the final result after the EIS completes the processing triggered by the
operation replay. There is no guarantee that the state of the cache matches what is in the EIS. However, the trade-off for this slight window of inconsistency is increased efficiency and reduced loading on the EIS.

There are two ways that the latest data can made it back to the cache:
- The valid duration time for the cache expires, and the next access triggers retrieval.
- EIS uses a data notification change to alert the SUP server (cache) of the latest state.

4 PERFORMANCE AND DATA MOBILIZATION PATTERNS
4.1 DATA MOBILIZATION PATTERNS

There are similarities between distributed computing environments and data mobilization. Between consistency, availability, and partitioning, developers can generally only choose two of the three. In mobile computing, when you put data on a device, you are, in essence, selecting availability and partitioning. You can partition data so it can be made available even when there is no connectivity or if it is too expensive to maintain one between the devices and the enterprise. In other words, full consistency is sacrificed to operate in this distributed environment.

Mobilization of data involves retrieving data from various back-end systems and storing it in a cache from which it can be downloaded to devices on demand. However, in most cases, the cache is only loosely connected with the backend systems via the exposed EIS interface. Unlike a hardware cache, which snoops on the memory bus to maintain consistency, the loosely connected cache usually relies on a refresh mechanism, for example, pull notifications or polling for the latest data. This results in complexity and performance impact when increased consistency is desired between the data on the device and that of the enterprise. It is important to understand that this discussion refers degrees of consistency, rather than full consistency.

Eventual Consistency

“Eventual consistency” refers to the pattern of allowing clients to temporarily diverge from what is actually on the back-end system. For usability, clients may maintain changes on its local data store until an actual result is returned from the EIS. This also means that other clients may or may not see the update after the first client has synchronized. Any CUD operation is dispatched directly to the EIS and state of the affected entity is not immediately available in the platform cache. This discussion focuses only on synchronous operation replay, as the majority of back-end systems do not support asynchronous interactions. If the EIS rejects an operation, the failure is propagated back to the client, which then has the option of throwing away the pending changes or modifying them for resubmission. Eventual consistency means that there is no requirement to issue an immediate cache population/retrieval operation to the EIS, either for a particular record or via a bulk-load to fetch the all of the latest changes. This may seem unusual to users who used to thinking in terms of ACID. In looking at a variety of mobile applications and the use cases they support, this policy is not unusual. For an Enterprise Information System that supports selective load/retrieval, tighter consistency is an option, but it comes at a price—please refer to “Immediate Consistency” below. Many mobility use cases seem to be centered on pushing status, reporting back to the enterprise rather than performing mobile OLTP.

Immediate Consistency

Immediate consistency is not the same as full consistency with ACID properties. Data partitioning and availability requirements mean that it is not really feasible to offer full consistency. If your situation requires full consistency, an online solution to a centralized data store may offer a better solution. “Immediate consistency” means getting back the result (or a close approximation) immediately after operations complete. In relational data synchronization, by default, the download phase always returns the result of the transactions and operations that are sent in the upload.
To achieve immediate consistency, one of the following must be available:
• Create/Update/Delete operations against the back-end that return the data state of affected entities, along with the status of the operation
• Selective retrieve operations to return only the modified data state of affected entities
• A bulk-load cache population mechanism. Such a mechanism is very expensive and impractical when the cache is shared by multiple clients
• Supports write-through cache that temporarily updates the cache using parameters associated with the CUD operations. The content of this cache is only an approximation of what is eventually returned.

Immediate consistency is useful in some cases. For example, if a salesman changes a customer’s city, the back-end EIS also updates the postal code based on the new information. With immediately consistency, the new postal code is returned in the download. However, it is a trade-off point for developers to decide whether performance or immediacy is more important.

Data Change Notifications
Sybase supports both push and pull interactions with back-end systems. In push interaction, the back-end system pushes change notifications to the platform. This does not give immediate consistency, but it has the potential of narrowing the window of inconsistency between the device and the back-end data source.

4.2 CONSIDERATIONS DURING DESIGN/DEVELOPMENT
It is important that you understand the use cases that are supported by the mobile application, and define the cache population model of the MBO accordingly. Failure to do so results in contention and interlocking of operations when applications synchronize with back-end systems through the platform. The following sections are general guidelines to help determine which cache population model to consider for a particular scenario.

Private Data Set with CUD Operations and Immediate Consistency
In this scenario, the user is performing CUD operations against his or her data set and wants to retrieve the result in the same synchronization. MBOs that require these characteristics should leverage the partition cache population model to reduce the overhead of retrieval operation and differential calculation.

Public Data Set with CUD Operations and Immediate Consistency
In this scenario, the size of the public data set is critical. The larger the data set is, the more work the server must perform (and it is likely to be nonproductive work) to maintain the cache and differential calculation. Heavy concurrent updates are likely to be not very scalable. If possible, partition the data set into shared units of work, for example, all sales orders that belong to a single sales team. Consider the partition model for cache population for this scenario.

Public Data Set with Mostly Read Data
This scenario comprises mostly read data, for example, products that are shared among many users. The bulk-load cache population model makes the most sense. Turn on caching to avoid hitting the back-end system on every access, thus making the system totally non-scalable. Also, use scheduled refresh to avoid delaying the first access after cache expiration. This is particularly important for a very large data set. You may want to refresh the product MBO during the night, as it may require retrieving and comparing thousands of record to detect changes. During the download, which may take place the following morning, only changed items are sent to the device.