Tamino Mobile

XML Database

TECHNICAL WHITE PAPER

SOFTWARE AG
THE XML COMPANY

Contents

Software AG
September 12, 2001

Introduction

Tamino Mobile 3.0 is a portable, small-footprint database engine tailored for and targeted at the rapidly growing market of handheld devices and embedded systems. Location-independent work environments and pervasive computing require solutions where both structured and unstructured information is recorded, stored and transferred across a multitude of computing equipment and devices, and across a variety of communication lines.

With a footprint ranging from 400k-700kB of required memory, Tamino Mobile is designed for the particular requirements of devices with limited processing power, memory and battery capacity. It is also designed for wireless communication with limited bandwidth. Although technological advances allow these limitations to be continuously reduced, resource use will always be an issue. With Tamino Mobile, centrally maintained data can easily be synchronized for offline access in environments where direct wireless connection is not possible or economical.

Tamino Mobile allows storage, search and navigation within a data collection. It provides an abstract model of the data, as well as programmable operations and measures that ensure data integrity when several tasks or users operate on the same data. Tamino Mobile combines powerful mechanisms to meet programmers' expectations, with a simple architecture adapted to the limitations of handheld devices and embedded systems.

The database may run either as a pure in-memory database or as a database with secondary storage, such as flash and micro-disk. Because of its modular architecture, it can be optimized for specific application areas, balancing small footprint, performance, robustness and programming features.

Tamino Mobile in a Nutshell

Today's solutions for mobile end user devices are designed mostly for offline use. PDAs and handheld PCs are the devices of choice. However, with the introduction of new mobile transmission technologies, such as WAP, UMTS, GPRS, Bluetooth and the like, a rapidly growing number of users will opt for online systems.

While offline systems tend to be based on applications installed on stationary systems such as desktop PCs, the new online systems generally rely on a browser-based architecture. By virtue of specifically formatted pages (WML, HTML, cHTML) the browser provides quick and easy access to server-based applications. This stands in contrast to offline systems, whereby synchronization with a central server system – usually desktop systems only – is provided through hard- and software solutions.

Developers creating new applications will first have to decide which kind of system they will go for: online or offline? The fulfillment of user requirements often suffers due to technical limitations, which is the reason why many mobile solutions have not been enthusiastically received by users.

Ideally, any new type of mobile system must combine the criteria for online and offline solutions.
That's exactly what MobileLogic does.

MobileLogic is Software AG's professional services framework, enabling developers to quickly generate custom and browser-based applications. Tamino Mobile can be used in combination with Software AG's powerful Tamino XML Server. While data will typically be maintained on a centrally located server, desired information must be synchronized regularly for instant offline access through handheld devices, etc.

MobileLogic provides pre-developed components necessary to synchronize Tamino Mobile's database content with centrally maintained information, as well as querying and displaying this data offline on the PDA screen.

The following table provides an overview of the main selection criteria:

Criteria for an online solution	Criteria for an offline solution
• Currentness of the data • Simplified administration of the application • No installation necessary on the end user device • Interactivity – e.g. with other users	• Independence and availability of mobile networks • Low communication cost • Independence of user load on servers • Individuality

A data loader module is included for down- and uploading content from a remote Tamino XML Server into Tamino Mobile's database repository. Synchronization is either executed on request or carried out automatically after being triggered by server-based monitoring functions. Through a server-side portal you can define which server data is available for download onto the mobile database. This significantly reduces the transfer time due to the lower volume of transmitted data, shrinking transmission cost dramatically. For synchronization with the central XML server, the PDA can be connected via the corporate LAN, wireless LAN, GPRS or UMTS.

The complete framework is depicted in the following diagram.

Tamino Mobile is built to enable easy customization in order to satisfy a specific set of requirements. Software AG knows that demands placed on a database differ depending on the platform, the hardware resources, the characteristics of the application, the available development tools, etc. We are also aware of conflicting requirements, often pulling the user in different directions.

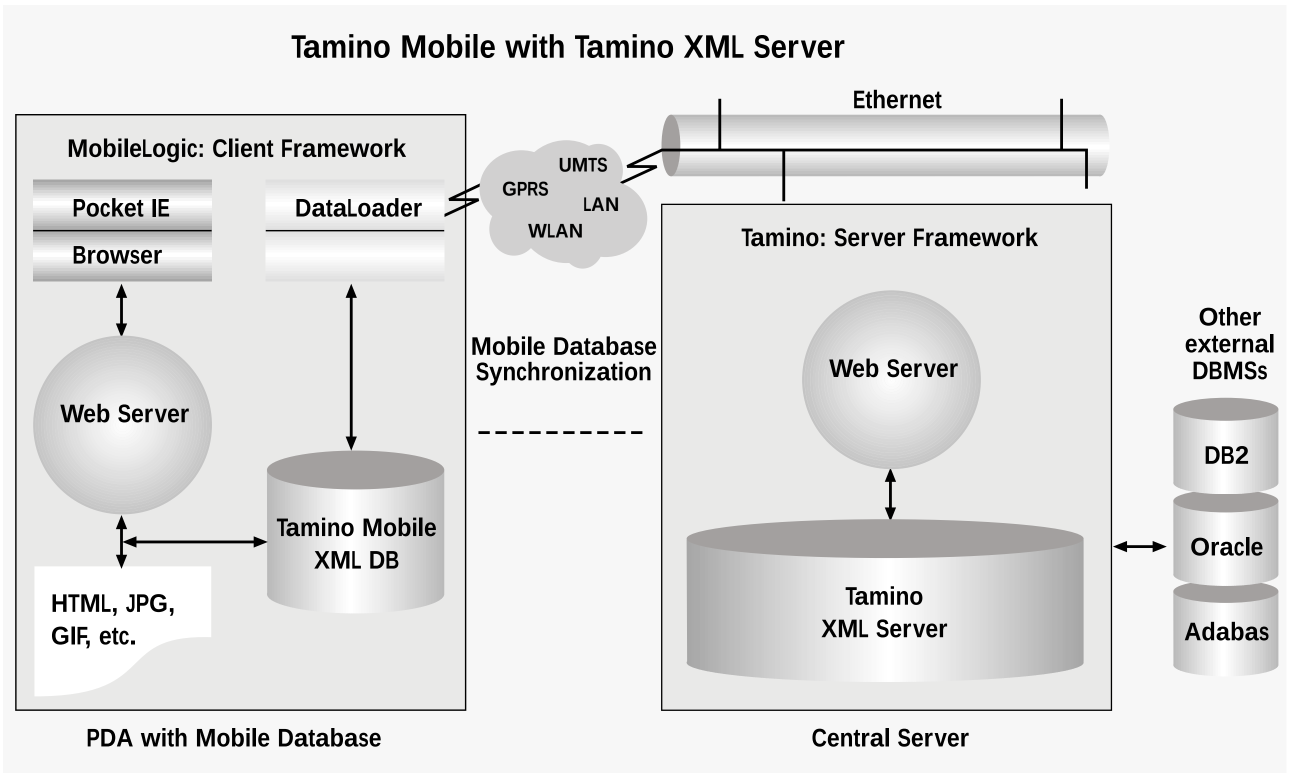

Fig. 1: Synchronization of Tamino Mobile by virtue of the MobileLogic framework

Tamino Mobile - A Customizable Database

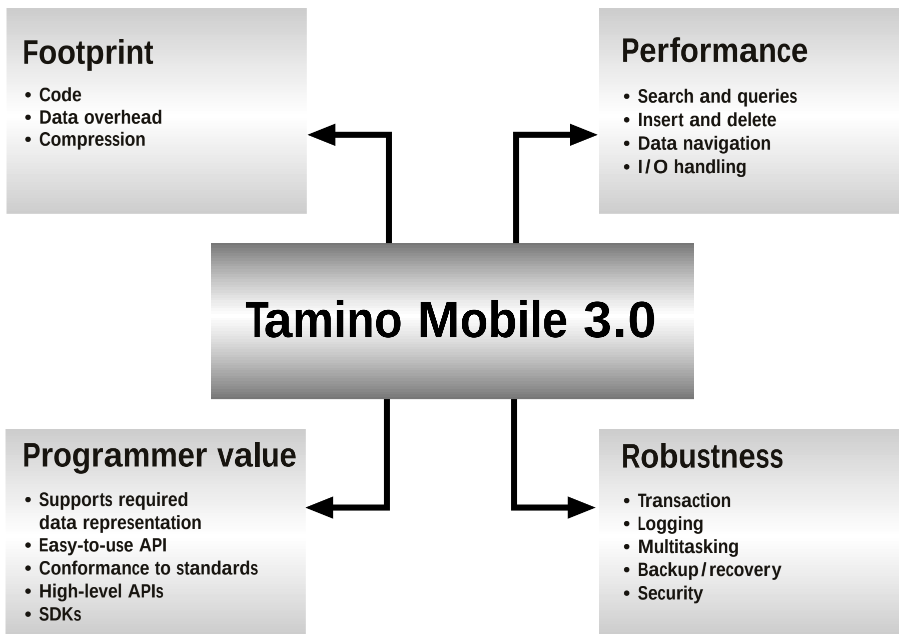

Fig. 2:
Typical requirements for mobile databases

The primary aspects that need to be balanced for an optimal mobile database implementation are:
• Footprint
• Performance
• Robustness
• Programmer value

Figure 2 illustrates typical requirements in these four areas.

For optimal database performance, customization capabilities are provided on two levels:

1. Installation parameter setting

When installing a Tamino Mobile database it is possible to set parameters and tune the database, e.g., to balance performance vs. data footprint.

2. Programmable mechanisms

You specify your requirements and Software AG's Professional Services team develops your custom Tamino Mobile application. Tamino Mobile is flexible in that it allows programmers to choose data structures and define indexes for speeding up data access, navigation and searches.

Components

The Tamino Mobile database engine consists of the components shown in figure 3.
Let us go through them in more detail:

Tamino Mobile API
The Tamino Mobile API (the "native" API) is a set of methods encapsulated in classes. These methods constitute the programmable interface to be used by application programmers and by Software AG when in higher level APIs, such as XML-oriented applications.
Note: The XML API is not discussed in this white paper. For more details refer to the "Tamino Mobile XML APIs" white paper.

Database Schema
The database schema contains meta information about the data stored in the database, i.e. information about the structure and types of the data objects and attributes in the database. The database schema is updated either through explicit calls from a program, or by importing database schema information in a prescribed format.

Query Engine
The query engine performs query operations returning collections of objects matching the selected criteria. The query engine also provides a cursor mechanism allowing the application programmer to iterate through the collection.

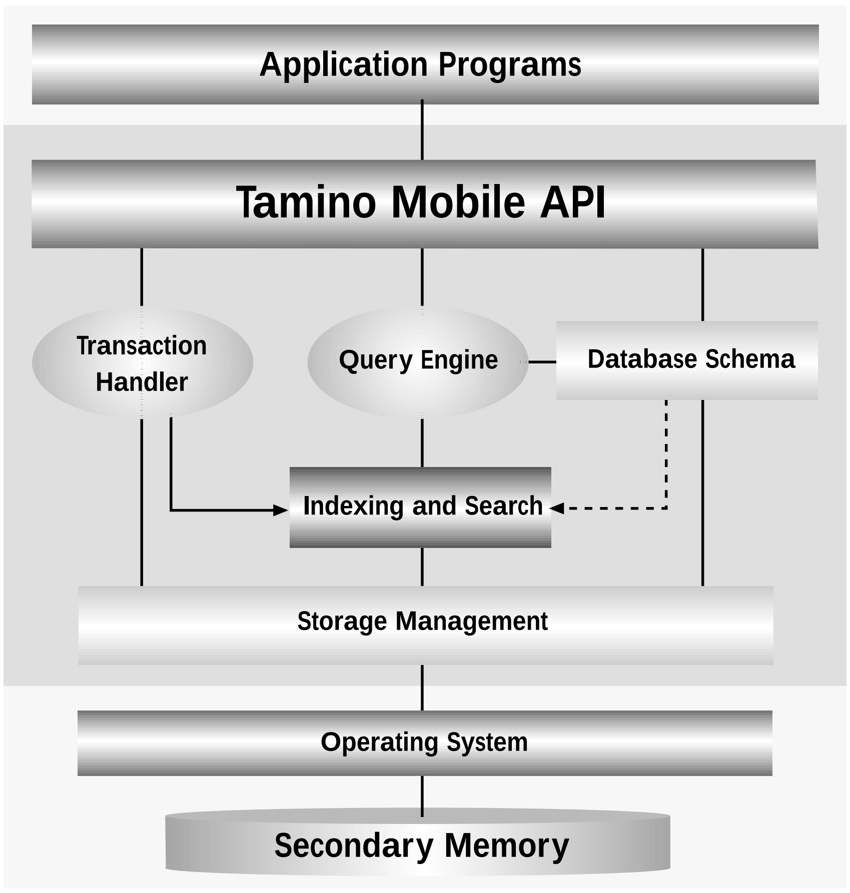

Fig. 3: Components of the Tamino Mobile database

Transaction Handler
The transaction handler is a service for the application programmer to ensure that any write operation is done exclusively. Many read operations can be processed in parallel, but they will all have to terminate their transactions before a write operation is allowed to start.

Indexing and Search
This module includes mechanisms for building indices when data is entered, deleted or updated and contains fast search routines in subsequent lookup routines.

Storage Management
Storage management organizes the creation, deletion, modification and retrieval of objects and their attributes. It also administrates memory allocation using file- and memory-oriented features of the underlying operating system platform. The storage management module is not directly accessible from an application program, but is accessed through the database schema, query engine and the transaction handler.

Data Structures

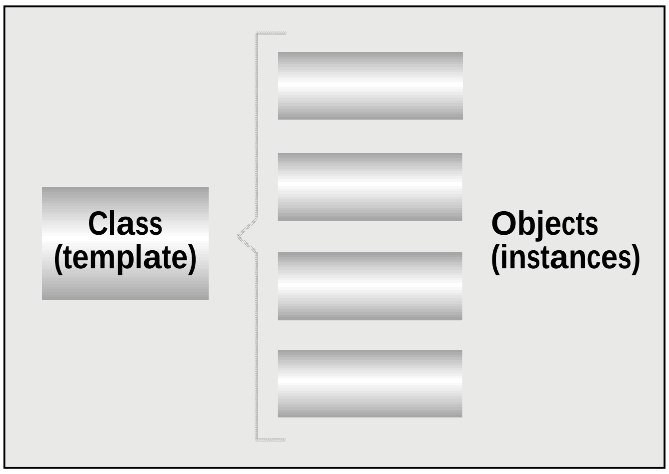

Fig. 4: Relationship between objects and classes

Objects
A Tamino Mobile database populated with data can be considered a collection of objects. Each object must have a type (class) which has been declared in the database schema.

Every object has an object ID which is unique among all the objects in the database. The object ID is assigned by the database system when the object is created, and is not reused for new objects once the object has been deleted. Thus the object ID can be used to identify the object without risking that the identity has been taken over by some other object.

Attributes
An object may have zero or more attributes. An attribute is either mandatory or optional. Optional attributes may be dynamically created or deleted. For objects with only optional attributes, a corresponding class is not needed. Tamino Mobile supports simple inheritance, meaning that a class can be declared as a subclass of a previously declared class (superclass), the effect being that all the attributes declared in the superclass at the time are automatically part of the subclass as well. Subclassing can be done to any desired level.

Optional attributes declared:

- An object can have zero or more attributes.
- Attributes declared in the database schema must be declared with an attribute ID (of unsigned integer type).
- Each attribute has a value which is read/written atomically. The value is a built-in type such as integer, text, object identifier, etc.
- An attribute is either mandatory or optional. Classes may be declared to not support optional attributes in order to optimize storage utilization.
- All attributes have an attribute name that is unique among the attributes of the class (including inherited attributes). All mandatory attributes are declared with name and type in the database schema. Optional attributes do not have to be declared in the database schema. Their name and type can be given when the attribute is created.

The database system will group mandatory attributes of the same object in the database storage system in order to optimize storage efficiency and spatial locality of reference to the underlying storage medium. Instances of optional attributes may be created and destroyed dynamically. Optional attributes are therefore allocated in a linked list with its header in the object. By optionally restricting an object to have only mandatory attributes, this header is no longer needed in the objects, and storage utilization can be improved at the expense of the flexibility of the application data model.

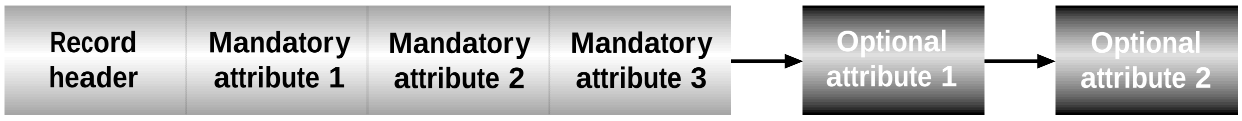

Fig. 5: Object with mandatory and optional attributes

Data types

The following are types of object attributes:

- **UChar** represents an unsigned single-byte character
- **Char** represents a single byte
- **Ushort** represents an unsigned short integer (16 bits)
- **Short** represents a signed short integer (16 bits)
- **Ulong** represents an unsigned long integer (32 bits)
- **Long** represents a signed long integer (32 bits)
- **Float** represents a signed floating-point number of 32 bits
- **Double** represents a signed floating-point number of 64 bits
- **Text** consists of a string of ANSI text characters
- **Unicode** represents a 16-bit character. Unicode contains 49,194 distinct coded characters.
- **Blob*** is an acronym for Binary Large OBject, and is used to hold binary coded contents such as sound, image, animation, etc. It can also be used as the attribute type of any data where operations and semantics are irrelevant.
- **Object ID** is a reference to an object in the database
- **Time** represents time
- **Date** represents date

* Note: BLOBs (Binary Large OBjects) are typically used to implement multimedia elements, such as sound, pictures, animations, moving images, etc. They generally consist of large chunks of data, and their implementation in Tamino Mobile has therefore been realized in a unique way to avoid performance and storage overhead. BLOBs are divided into blocks of CPU page size and aligned to minimize the number of page faults when accessing the database file. A Blob is stored as a page directory, listing the pages that make up the BLOB. BLOBs may be accessed in two ways, either through ordinary read and write calls, which invoke a copy operation, or through memory mappings that can be established on a 'per block' basis. When using memory mappings, copying is avoided. Instead, the mapping is passed on to the application program. Copying will then only occur if the application program chooses to do so.

Semistructured Data

Tamino Mobile supports semistructured data. This means that there is not necessarily a database schema that describes the structure of the entire content of the database. In order to offer good support for storing XML documents and similar data with a less fixed structure, Tamino Mobile may contain data that is not covered by the database schema.

The following mechanisms support the representation of semistructured data:

- Each object may be associated with a type (class) which has been declared in the database schema.
- A class can have a combination of mandatory attributes and optional attributes.
- Optional attributes can be added dynamically, enabling representation of data structures that are not defined initially through the database schema.
- Objects can also be created with out an association to a declared class. In such cases, the object has no type, and the object has no mandatory attributes. It has only optional attributes whose types will be given as the attributes are created.

Hierarchical Data Structures

Objects in a Tamino Mobile database may be organized in a hierarchical structure. Therefore, Tamino Mobile lends itself to representing XML-oriented, hierarchical and graph-oriented data structures. Tamino Mobile contains mechanisms to establish the structure and navigate within the structure.

The following features are available to programmers:

- An object may have zero or more objects as children.
- The children of an object are ordered. When new children are created, they can be inserted relative to some other child.
- An object has at most a single-parent object.
- The edge that connects an object to its parent object has a textual label that describes the role of the object in its parent.
- Labels can be combined to form paths in the database. Paths can be used to limit the universe in queries on hierarchically structured data.
- Objects with the same parent may have the same label on the edge that connects them to their common parent. This means that a path specifies a set of objects, not a single object. Another way to look at this is to view a label as a subdirectory name, and children attached to the same parent through the same label as part of the same subdirectory. The label represents a subdirectory, and objects can be viewed as files stored in that subdirectory.

• Graphs can be represented through a combination of parent-child structures and through the use of attributes referencing other objects in the database.

Database Schema

The database schema contains information about the structure of the data objects and attributes in the database. The database schema is updated either through explicit calls from a program, or through importing database schema information in a prescribed format. The database schema typically holds information about the names and structure of classes (object templates) and the name and type of attributes. It also holds information about indexing.

All attributes are declared with name and type (integer, text, etc) in the database schema. The database schema for a class can be extended in order to accommodate further optional attributes dynamically. Note that while the database schema can be extended, there are no methods that can modify the schema in such a way that a database instance valid under one schema will become invalid under the modified schema. Thus, we can allow an application to modify the database schema, knowing that the application cannot modify the database schema, so that the existing database instance would violate the modified schema.

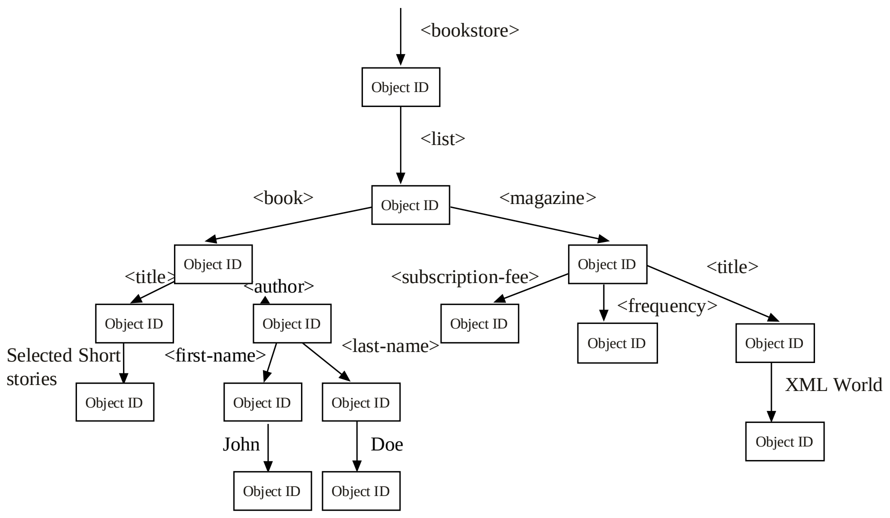

Fig. 6: Hierarchical object structure - example from XML

Accessing Data

All access to data in the database is done in three steps:

i) Define the collection of objects to be accessed

A collection is a subset of the objects in the database and is defined dynamically in one of the following ways:

• Queries. The collection consists of all objects matching a given query. The result of the query will return a number of objects (or none if there is no match).
• Navigating in the hierarchy. Navigating up towards the root will return one object, the parent. Navigating down towards the leaves will return a number of objects (or none if there are no children).
• Creating a new object. The collection consists of the new object only.

ii) Locate the object within the collection, if more than one, using cursor iterations

Every collection has a cursor, which is a reference to the current object in the collection. For collections with several objects the cursor is used to iterate the collection. The database provides mechanisms for this iteration, such as GetFirst, GetNext, etc. For singleton collections, i.e. collections which cannot have more than one object (typically the result of finding a parent or creating a new object), iterations on the cursor are ignored.

iii) Access the attribute(s) of the object

Whenever an object is made current, its data may be accessed through mechanisms such as Read, Write, etc.

Indexing

The support for both structured and semistructured data has consequences for how indexing is done by Tamino Mobile. Indices can be regarded as part of the database schema because they are very important for the practical use and the tuning of database performance.

Structured data (mandatory attributes) can be indexed using three indexing mechanisms:
• single-attribute indexing
• multi-attribute indexing and
• attribute-group indexing.
All mechanisms invoke index-building routines when data is entered, deleted or updated, and invoke fast search routines in subsequent lookup routines.

Single-attribute indexing causes all existing values of the given attribute to be entered into a separate search structure for that attribute only.

Multi-attribute indexing defines two or more attributes within objects of the same class to act as an ordered tuple, i.e. as an ordered sequence of values. All tuples, i.e. all existing combinations of values, are entered into a separate search structure. Attributes that are part of multi-attribute indexing must be mandatory.

Attribute-group indexing defines two or more attributes within objects of any class to belong to the same search structure. This means that all values of attributes of the attribute group will be entered into a separate search structure for that attribute group only. For obvious reasons all attributes must be part of the same base type. Attributes in an attribute group index may either be mandatory or optional. This allows a program to perform a search that spans across different attributes.

This is very useful when searching semistructured data.
When accessing indexes, the database engine uses iterative algorithms. Thus, there is an upper limit on the amount of stack space used.

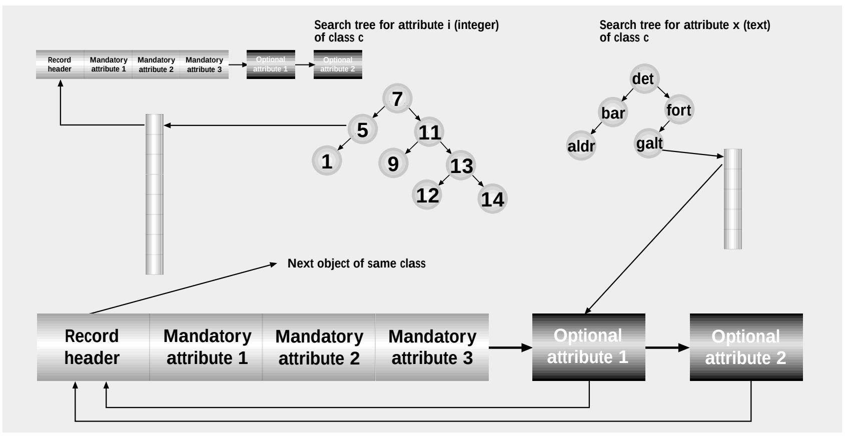

Fig. 7: Single-attribute indexing

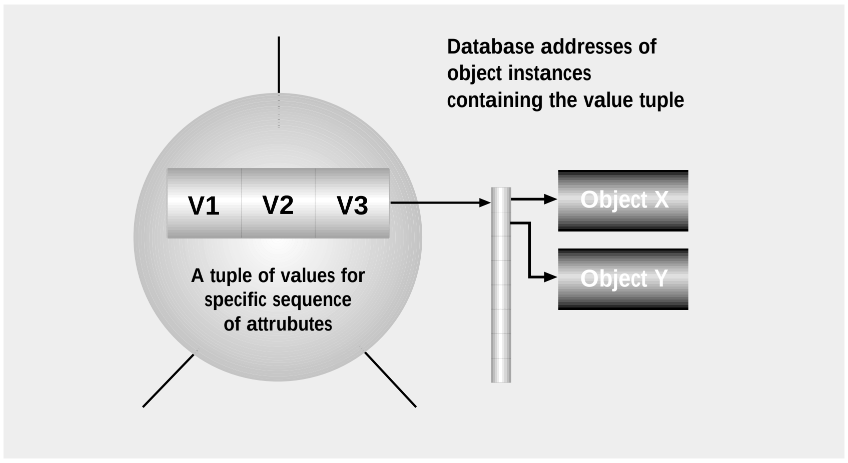
Database addresses of object instances containing the value tuple
V1
V2
V3
A tuple of values for specific sequence of attrubutes
Object X
Object Y

Fig. 8: Multi-attribute indexing

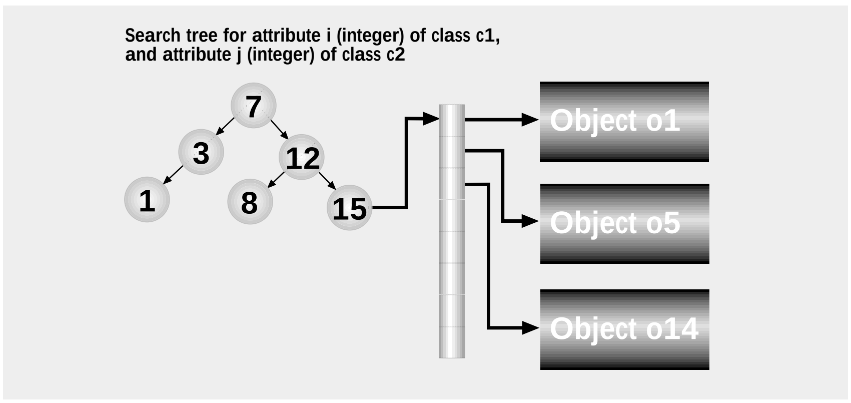
Search tree for attribute i (integer) of class c1, and attribute j (integer) of class c2
7
3
12
1
8
15
Object o1
Object o5
Object o14

Fig. 9: Attribute-group indexing

Queries

Queries in Tamino Mobile are performed using the Lookup functions LookupByClass and LookupBy-Group.

In LookupByClass the query will return all objects of a given class having attributes that match specified criteria. For each attribute/value pair, an operator may be specified. Attributes used in a LookupByClass must be mandatory attributes.

In LookupByGroup the query will return all objects of classes where the attribute belonging to the group matches the specified criteria. The collection of objects in the result set may consist of objects of different classes.

Multi-Threading

Tamino Mobile processes will not preempt other processes. Tamino Mobile can be run as an in-process library without creating any independent threads of control. Its database engine does not interfere with processing or other low level scheduling in the operating system.

Tamino Mobile 3.0 is based on a multi-threaded database that can easily handle multiple users. This is true whether it is different processes with different address spaces or different threads in the same process that use the same database. Database access from different processes in different address spaces is controlled through an OS-dependent semaphore that grants access to the database and to the processes one at a time. The semaphore is part of the specific implementation for each platform and might be realized differently on different platforms.

Secondary Storage

Tamino Mobile was designed to provide maximum flexibility with regard to the actual storage medium. This ensures that the database engine can be tailored to any storage medium. Various types of secondary storage are supported, such as traditional disks, flash memory, microdisks and memory sticks.

Tamino Mobile has its own database medium abstraction layer shielding the database core from the actual storage technology used by the device. Adapting Tamino Mobile to a device consists of implementing a version of the database medium abstraction layer for that platform in order to make the best use of the underlying storage hardware.

Tamino Mobile uses a storage allocation algorithm that does coalescing of adjacent free blocks. Also, the allocator maintains separate free-lists for each block size up to a certain size, thus preventing fragmentation by reusing blocks of the same size.

Transaction Mechanism

The transaction-handling scheme is based on a read-many, write-one strategy. This means that all readers must complete their transactions before the writer is allowed to start. The writer will then have exclusive access to the database. This transaction scheme is very simple to implement and is also very effective. It is the responsibility of client applications to begin and end code segments with BeginTransaction and EndTransaction tags.

Note: It is possible for a transaction to start as a read-only transaction and later upgrade to a write transaction.
Traditional database designs go to great lengths to allow as many transactions as possible to proceed concurrently. This is important in order to maximize the utilization of the disk channel and cache:
While one transaction is blocked waiting for the disk I/O to complete, other transactions may be able to run using data from the disk cache and should be allowed to execute concurrently with the blocked transaction.

Tamino Mobile will typically run as a single-user database. Also, some devices have no secondary storage, so that the database will run as an in-memory database. This means that there will be no or very little I/O. Only seldom is a transaction blocked for an I/O to complete. The durability property of transactions obviously requires that I/Os take place at the end of a transaction, but this requirement may be lifted on certain platforms where there is little to gain by allowing concurrent execution of transactions on the database. On such platforms Tamino Mobile will implement the ACID properties by executing update-transactions in a serial fashion, thereby eliminating the need for complex locking tables, etc. Read-only transactions are allowed to proceed concurrently as long as there is no write transaction in progress. This simple transaction concurrency regime offers a practical approach to transaction concurrency and dramatically reduces the code footprint of the database engine.

Transactions executed by Tamino Mobile have the following properties, also known as ACID properties:

- Atomicity:
  A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all. If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery method must undo any partial effects of the transaction on the database.

- Consistency preservation:
  A correct execution of the transaction must take the database from one consistent state to another.

- Isolation:
  A transaction should not make its updates visible to other transactions until it is committed.

- Durability (persistency):
  Once a transaction changes, the database and the changes are committed, these changes must never be lost because of subsequent failure.

All in all, this guarantees serializability, meaning that the effect of running transactions in an interleaved fashion is equivalent to running them serially in some order.

Note: The transaction mechanisms are necessary only when the device has secondary storage capacity. If a device, such as a PDA, is used with this product, there is no need for transaction mechanisms. When the device loses all power, then it resets to the factory settings, thus all software must be reinstalled. A PDA can be turned off in middle of a transaction without affecting the database, since it operates in such a way that it does not kill the transaction. But, if the device requires extra memory, such as flash cards, etc, and these are removed in the middle of a database update, then transaction mechanisms are necessary to ensure the consistency of the data. It is in this case only that you require such mechanisms to ensure data consistency. For this purpose, Version 3.1 will support such mechanisms.

Transaction Log[1]

A writeback cache will be employed to delay the writing of database updates until the transaction commits. Whenever something is evicted from the cache and must be written to the persistent medium, the overwritten data is first flushed to a transaction log with a marker at the end. Only thereafter will the database medium be updated with the data evicted from the writeback cache. When the transaction ends, the transaction log is deleted. If a transaction log is found to be present when the database is opened, the log is carried out on the database until the last marker in the log appeared, in order to reverse the effect of the transaction that was only partially completed.
A transaction in Tamino Mobile is either carried out in its entirety on the database or has no effect at all, even when faced with a loss of power. This transaction atomicity is guaranteed by introducing a transaction log and maintaining the following invariant:

The database file and the transaction log taken together will always reflect the database state at the end of the last committed transaction. To uphold this invariant, the database engine needs to write the transaction log to persistent storage whenever something is evicted from the cache or a database transaction is committed. The transaction log has to be stored persistently before Tamino Mobile can proceed to update the database file. The transaction log is typically short-lived and is frequently accessed. Tamino Mobile can therefore be configured to place the transaction log in NVRAM, if such a feature is available on the hardware platform.

Tamino Mobile can limit the size of the transaction log for the entire database at compile time if the customer wants to introduce an upper threshold for all applications that use the database. The database engine would then abort all transactions that exceed the size of the transaction log. This will probably only be used in embedded systems with limited hardware capabilities or strict footprint requirements for user data.

[1] Transaction logs will be supported starting in October 2001. Tamino Mobile Versions 3.1 and higher, available on Win32 and WinCE, will support this feature.

On many PDA platforms, switching off the PDA is akin to a task switch. When power is switched off, the PDA will enter sleep mode until power is turned back on. The PDA processor will therefore terminate any ongoing transaction before it is switched off. On such platforms, there is no need for a transaction recovery mechanism. This minimizes the code footprint even further.

On non-PDA platforms, the portability layer of Tamino Mobile will implement the proper recovery mechanisms at the price of somewhat higher code footprint.

Implementation and Portability Issues

Tamino Mobile is implemented in C++ with portability layers that can adapt the database to any hardware platform. The implementation utilizes object-oriented mechanisms. This leads to a concise and well-defined API, and simplifies application development. Moving to a new processor will require only a recompilation of the Tamino Mobile source code, while moving to a new operating system and/or hardware will require minor changes in the portability layer. There are no #ifdefs in the Tamino Mobile 3.0 code. Instead we have defined a portability layer, which is given a concrete implementation for each platform we port to. The rest of the source code rests on top of the portability layer and does not have to be touched when porting. The portability layer handles all adaptation to a certain environment.

The database image is binary-compatible across operating systems, but may require conversion by a standard conversion tool when moving between platforms having different byte ordering. The requirement to run a conversion tool can be removed at the expense of performance (in this case Tamino Mobile will perform byte order conversion at runtime).

Tamino Mobile runtime versions are available for Windows CE 3.0 on handheld PCs (StrongARM, MIPS or SH3 processors supported), and further runtime licenses will be available for Windows ME, Windows98. Runtime versions for other platforms can be made available on request.

Tamino Mobile does not use any C++ libraries. The code is completely self-contained. This includes memory allocation, where new() and delete() operators have been overridden to use the memory allocation of the underlying operating system. For specific operating systems, we have developed our own memory allocation mechanisms.

Tamino Mobile can use standard C++, such as libc, instead of its own libraries, if necessary.

MobileLogic – The Mobile Service Platform

Technical Description

MobileLogic is primarily designed as an open solution, specifically regarding integration aspects and extensibility. Thus, it guarantees a maximum of interoperability with application servers, object request brokers, Web services and standard applications. Throughout the entire processing chain XML is used as the common data description language and for interoperable method invocation. This turns MobileLogic into a powerful, universal instrument for mobile access to existing and to-be-developed services.

COMPONENTS:

Service Integration Framework:
The service integration framework enables the integration of any kind of service provider, such as Web services, components, application and security systems through a unified XML interface.

Content Cache:
In principle, the informational content of any message is time-related. The content cache ensures that not every system request results in immediate accesses to systems running in the background. In effect, system scalability is significantly improved.

Meta Data:
Meta data contains all system information necessary for operation. This includes, for instance, personalization data, user preferences and a directory for all available services.

PDA Mirror:
The PDA mirror provides all data that the mobile end device has been synchronized with. On one hand, this guarantees that all data residing in the mobile end device can be restored in case of a system failure (e.g. battery power loss). On the other hand, changes can be detected instantly, allowing for automatic data exchange / update between connected systems.

MobileLogic – The Mobile Service Platform

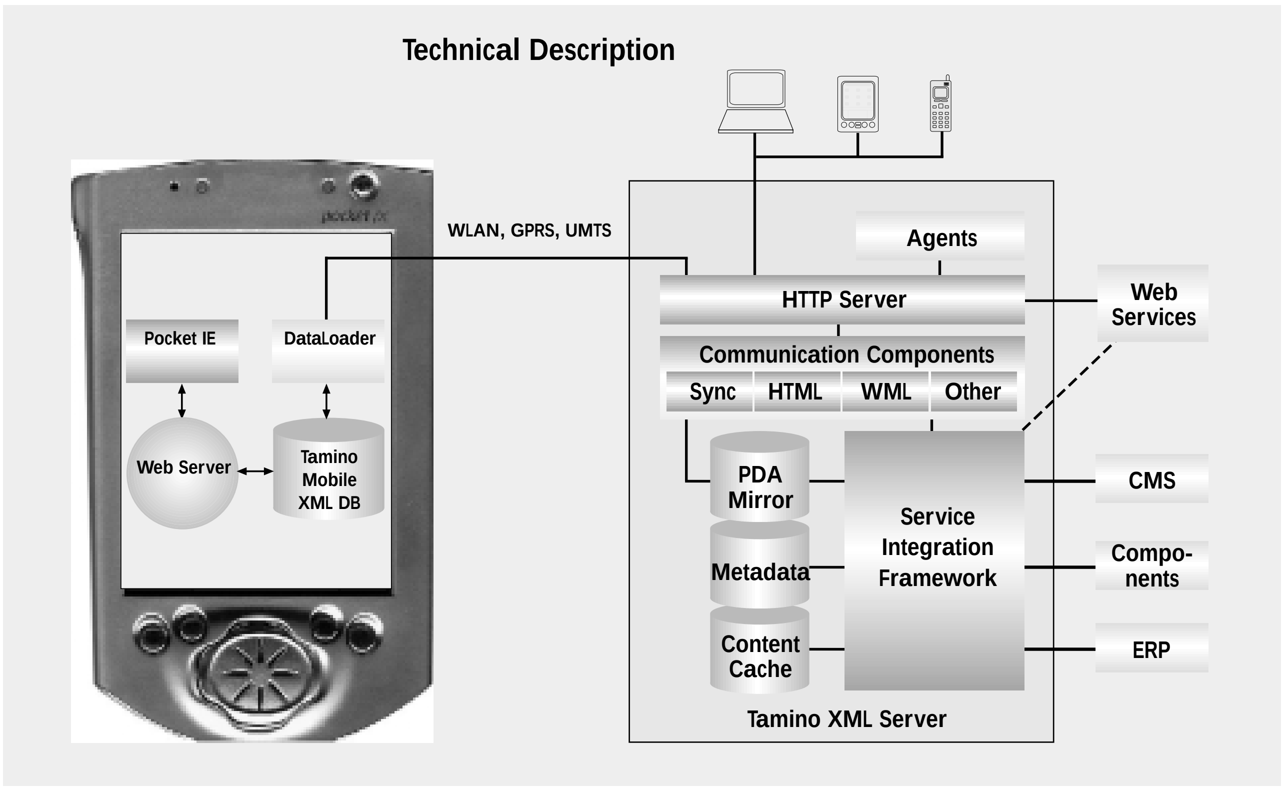

Fig. 10: Primary components of the MobileLogic development framework.

HTTP Server:
The HTTP server – or Web server – provides access to system resources using the standard Internet protocol HTTP.

Agents:
Agents are components used for controlling and monitoring the system. Depending on their purpose, they utilize user-specific data buried in the meta data. For example, notification of the user about new information received would be an example of a monitoring function. Subsequently, this information can then be synchronized manually or automatically.

Communication Components:
The access layer is the level of communication processing for the respective end-user device. It is implemented based on XML.
Access control is carried out by a system component specifically provided for the respective end device. Thus, for example, a Web component based on Java Server Pages is used for HTML-based browsers.
XSLT (stylesheets) provide for the transformation and representation of information objects.
A proprietary description of access modes, information representation and control (e.g. HTML page structures) by outsourced menu structures has generally been avoided, due to the complexity of such methods and the poor cost/benefit ratio, Thus, the control of voice modules cannot be described in the same manner as HTML pages, for example. In addition, it should be possible to generate a suitable user interface by using standard programming methods and tools. The communication components also implement the synchronization with the mobile end device. The network connection can be established using any IP-based method, such as wireless LAN (WLAN), GPRS, UMTS or interconnection via mobile GSM networks.

Data Loader:
The data loader is a component residing on the mobile end device. It controls the synchronization with the remote XML server. The synchronization rules are integrated in the meta data specified and managed on the XML server.

Tamino Mobile:
Tamino Mobile stores the XML documents (orders, bookings, news, etc.) as well as the associated stylesheets for accurate representation of the respective XML data (HTML). Since stylesheets must also be synchronized with the remote XML server, it is obviously the best choice to manage and maintain the data of the mobile system directly on the

server. This way, changes to the content can automatically be transmitted to the mobile end device. Besides XML-based data, it is possible to store pictures and audio (binary files) in Tamino Mobile. Thus, Tamino Mobile represents a unified, personalized database for all kinds of information.

Pocket IE:
The browser (Internet Explorer) is the central I/O interface for the user. It displays the HTML pages generated from XML documents and stylesheets by the PDA's Web server. It is easy to generate entire applications by just linking single information objects, overview pages and input forms. Links pointing to external information pages can be resolved in the same way as links to internal data stored in Tamino Mobile's database. Thus, for a positive user experience supported by standard technologies, the browser combines online and offline offerings with a consistent view.

PDA Web Server:
On the mobile end-user device, the Web server implements the interface between the browser and Tamino Mobile. Therefore, references to stylesheets and associated XML data will be resolved and merged into resulting HTML pages. Input data from data entry forms can be stored in Tamino Mobile.

Areas of Application

When analyzing application scenarios, the Tamino Mobile system component must always be considered in the context of complete customer solutions. The description of the following use cases focuses mainly on Tamino Mobile as an inherent part of the MobileLogic framework. But this special view does not exclude Tamino Mobile from being used in other possible application scenarios.

Typical use cases are found in all enterprises with one or more of the following requirements:
• Personalized portal
• Customer relationship management system
• Sales force / service communication linking
in order to take a modern and technically future-oriented route in e-business.

PERSONALIZED PORTAL
Here, all enterprises are addressed, intending to offer heterogeneous services over the Web, mixed with services of public interest, in order to achieve a higher visitor frequency and improve the website. By enhancing a company's services portfolio with external, value-adding services, the personalized portal will lower the barrier for customers to provide their personal data. This allows for 1:1 marketing.

An intranet application within a large enterprise with many different branch offices and departments is a good example of a personalized portal application. The personalized portal is one of the core components of MobileLogic. Its advantages are a requirement throughout the use cases described later.

Within a personalized portal, Tamino Mobile provides mobile access to required information, independent of an available online connection. Constant access to the latest information is guaranteed by synchronizing Tamino Mobile with the appropriate remote XML server.

CUSTOMER RELATIONSHIP MANAGEMENT SYSTEM
Here, enterprises are addressed who need to contact and inform their existing large customer base directly and who want their customers to have contact to each other as members of a greater community.

Target groups are companies that want to maintain their existing customer base by offering them more services. Since globalization and liberalization of the market lead to higher customer fluctuation, businesses like energy providers, telecommunication providers or mailing and delivery services will most likely experience the highest impact.

In addition, such a system is ideally suited for high-volume retailers of high-quality goods, such as computers or telephones (Nokia, Siemens, Alcatel, etc.).

Besides delivering dedicated information, the login makes the customer step out of the shadows. This increases the likeliness that he/she will bind himself/herself to the enterprise emotionally and will provide more information about personal preferences. Customer loyalty will be increased if personal assistants are used regularly ("Finally I've found someone who really cares about me ...").

Through Tamino Mobile, customers can control all information relevant to them without the high costs of communication. The server provides a precise overview about which services customers have requested and been provided. It is even possible to monitor the usage patterns on the mobile end-user device by implementing respective counters. The usage information can be transmitted easily whenever Tamino Mobile is re-synchronized with the central server.

SALES FORCE / SERVICE COMMUNICATION LINKING

This application is most relevant to enterprises running a mobile sales force or service organization. Besides scheduling data (trips), all customer relevant data will be downloaded onto the PDA. This way all important and relevant information can be made available to the field. Examples of such data are accounts receivables lists, past ordering preferences, price lists or product descriptions, drawings for service technicians and more.

Orders can be entered from anywhere and at any time. By synchronizing the mobile end-user device, the MobileLogic system can provide for automatic transfer of the data into ERP systems connected to the central server.

Typical applications are found in enterprises with

- sales forces requiring quick and autonomous decision making. Some examples are, for instance, representatives of the pharmaceutical or health care industries or those selling products to retailers. Since most of these deals are on the fly, thorough planning and information are of vital importance.
- technical service staff who maintain or repair heterogeneous systems. For the trip planning or ad hoc services on request, the worker will be supplied with a mobile end-user device providing all data that is relevant for the specific services. This proactive work style minimizes idle time and empty loads – factors which account for a considerable percentage of costs in these types of enterprises. Good examples are maintenance technicians for gas stations, cooling appliances, hardware service technicians and similar staff.

EMBEDDED SYSTEMS

Implementing Tamino Mobile within embedded systems can maximize the standardization of these systems. Hence, the XML data format can be utilized throughout all areas of electronic business, even when the final user interface is a refrigerator door, a radio, a vending machine or a fully automated navigation system.

Tamino Mobile is ideally suited for creating applications with significantly greater user friendliness compared with conventional solutions. Thanks to Tamino Mobile, these mobile applications can be implemented efficiently and with a maximum of standardization and openness.

More information on
Tamino XML Server:
http://www.softwareag.com/tamino

Tamino Community:
http://www.softwareag.com/developer

Download XML Starter Kit:
http://www.xmlstarterkit.com

Software AG
Corporate Headquarters
Uhlandstraße 12
64297 Darmstadt/Germany
Tel: +49-61 51-92-0
Fax: +49-61 51-92-11 91
www.softwareag.com

TMO/WP01E0901 228

SOFTWARE AG
THE XML COMPANY