Kabira college essay

KABIRA A) Fault tolerant runtime engine The Kabira Infrastructure Switch?? is the central host for all Kabira applications. It is a high performance software transaction-switch that provides the infrastructure for building and integrating applications, integrating networks, and mediating between data flows. It is a fully distributed platform that can be deployed across multiple integrated computer processors within a single physical node, as well as across multiple nodes. In addition, the Kabira Infrastructure Switch handles programmatic complexities associated with the development of software solutions. It frees programmers from the painstaking task of writing and rewriting the code required to provide services commonly needed by all applications, services such as failure detection, transaction recovery, threading and concurrency, guaranteed messaging, online upgrade support, and object distribution.

Programmers can focus their attention on business requirements and on creating the models required to meet them. In addition to providing a rich set of services, the Kabira Infrastructure Switch has relatively small memory and disk resource requirements. This is because it stores all application data in shared memory, and leverages new operating system services such as threads and other shared memory services, that replace functionality traditionally provided by middleware and database servers. In addition it is designed specifically for objects created by Kabira’s automated code generator, so the two can work in tandem to provide an optimal implementation of user models. Finally, Kabira application functionality is developed simultaneously on UNIX (Solaris, HPUX) and Windows NT to leverage the strengths of each platform.

This eliminates the problem of products originally targeted for one platform that never run as well on a secondary platform. B) Model-based development environment All Kabira applications are designed using either third party graphical object modeling tools based on UML, such as Rational Rose, or textual representations of models. Using an object modeling methodology for distributed applications allows a business modeler to concentrate on the functional requirements of the application, not low-level implementation details handled by the Kabira Infrastructure Switch. The advantages of this approach become increasingly obvious when integrating a large number of different systems, each of which has its own APIs, message formats, and protocols. Kabira Infrastructure Switch represents all external adapters as objects in the modeling tool, and differences are hidden from the application developer. That means designers do not need to be experts in the external systems being integrated such as CORBA, SNMP, Java EJB, SS7, and so on.

Designers can construct an application using adapters already provided by Kabira and its partners. C) Fully automated code generationKabira code generation technologies support complete translation of object models into executable code that runs on the Kabira Infrastructure Switch. 100% of the code required to execute the object model is generated. The generator produces C++ code that is compiled and linked with libraries provided by Kabira and partners to create an executable program.

In addition to the C++ code generator provided to implement object models, Kabira provides code generators to integrate external systems with the runtime transaction-switch. Examples include: ?? Relational databases ?? CORBA clients and servers ? SNMP agent generation ?? Java clients and EJB

servers The adapter code generators map between the different object and data models supported by all of these adapters. This solves the “ impedance” mismatch problem when building applications that integrate different object and data models. Solving this problem eliminates the need for additional hand coding.

Example mappings include: ?? An object model and a relational data model ?? ASN. 1 and IDL ?? Object models to HTML ?? Java to Kabira’s runtime transaction-switch Because adapters are part of the model, they can generate to multiple latforms. D) Adapters Kabira and its partners provide a set of pre-built adapters. These adapters are available within the object-modeling tool as objects for the application designer.

The code to integrate them into an executable application is automatically generated. This includes any external definitions required to access these objects, such as ASN. 1 to define application specific MIB’s to a system management console, or SQL DDL definitions to store persistent object data. Kabira adapters can be used in any combination to support different application requirements. This solves the “ n-squared” problem of writing custom code to connect every pair of applications being integrated. For example, support for SS7/TCAP allows the building of telecommunications applications that participate in telephone call processing with open administrative adapters using SNMP to existing O.

S. ’s. The data collected by these applications must be made with a database. Because the adapters can be used in any combination, Kabira is useful for developing applications that reside on an adjunct processor in a

hardware telecommunications switch, SCP, or O. S. In cases where a pre-built adapter does not exist, developers can easily integrate new adapters into the Kabira runtime transaction-switch.

To do this, they first define an adapter model for an external API, which is an interface with operations and states. Implementation is done using a combination of model action language and inline C and C++ code to invoke the external API. The code that calls the external API is put directly into the modeled operations and states, that is, adapter “ glue code” is contained in the model but encapsulated in objects to separate it from the rest of the application. III) Product Requirements From experience with issues defined in the problem statement, Kabira identified the set of minimum requirements for Next Generation Infrastructure software. Performance levels able to support high volume applications Transparent high availability / robustness and manageability High productivity in development and integration These goals served as the requirements for the Kabira development team back in 1996 when they set out to create a new class of infrastructure software aimed at meeting the needs of the next millennium.

The Kabira Infrastructure Switch is the first platform that addresses each of the key requirements for the development and integration of new leading edge mission-critical systems. Kabira is today’s best choice for creating flexible n-tier architectures that avoid unacceptable complexity, simplify implementation, and support modification of deployed applications in dynamic markets. A) Performance Performance is evaluated using various metrics. All metrics must be addressed by application platforms to allow deployment of highly available and reliable applications.

Key performance metrics include: Scalability Scalability allows an application platform and hosted applications to grow with increased application requirements by adding hardware resources. These hardware resources may be additions to existing systems or entirely new systems connected via a network. Throughput Throughput is a measure of the total units of work that can be processed in a specified unit of time. Traditional database servers use transactions to measure total throughput. However, this is only part of the total work that an application must perform to provide a service. LatencyLatency is a measure of cycle time required to complete an application-defined unit of work.

It is measured from the start of the work to its completion. Predictability Predictability is the ability to determine the runtime characteristics of a system before the system is deployed in a production environment based on application characteristics and load The Kabira Infrastructure Switch specifically addresses each of these dimensions of performance in its architecture. This provides the application designer the flexibility to control performance trade-offs to meet application specific requirements. For example, a designer might choose to sacrifice latency (by minimizing data caching in the Kabira Infrastructure Switch) in order to minimize the hardware resources required by an application, such as real memory and disk. In contrast, traditional middleware and back-office servers, such as databases, have addressed only one or two of these performance areas and in some cases they have done so only partially.

For example, while database servers scale effectively on a single hardware platform, they do not provide transparent scaling across multiple hardware

nodes connected over a network. i) ScalabilityThe Kabira Infrastructure Switch is designed to scale on both a single platform and across multiple platforms. Applications hosted on the Kabira Infrastructure Switch can be deployed on large multiprocessor machines or many smaller machines to meet application performance requirements. Supporting both single and distributed scaling mechanisms provides the application designer with the flexibility to make trade-offs between cost, manageability, and redundancy as required. Platform Scaling The Kabira Infrastructure Switch is designed to take advantage of the CPU, memory, and disk resources available on its host platform. As CPUs, real memory, and disk space increase, the Kabira runtime will scale transparently to the application.

The Kabira Infrastructure Switch achieves CPU scaling through the use of operating system level threading. The total number of threads used by Kabira is optimized to minimize “ empty CPU cycles” caused by excessive thread context switching. This ensures that CPUs are kept busy performing application work, not operating system housekeeping. The architecture does not just “ throw threads at the problem” to simplify the implementation.

In addition, the Kabira Infrastructure Switch does not perform global locking. All locking of shared resources is designed to minimize lock contention. This is accomplished by eliminating global resources that must be locked by all threads before any work can be performed. The Kabira Infrastructure Switch uses shared memory to achieve scaling of memory and disk resources, to provide a fast form of persistence, and to eliminate IPC overhead. Specifically, it stores all application data in shared memory, and then maps it to files using operating system primitives. This allows the operating system

to swap application data to disk if there is not enough real memory in the system.

As real memory is expanded, the shared memory in the Kabira Infrastructure Switch will take advantage of the added resource, and application performance will increase. Because shared memory is mapped to disk, it does not disappear when the application using it stops, providing a fast kind of persistence. Distributed System Scaling Application scaling can also be achieved by distributing application components across multiple machines in a network. These machines can be the same or different hardware architectures and operating systems.

Distributing applications on the Kabira Infrastructure Switch is done by partitioning models. This allows applications to be easily changed from a one node to many nodes, and requires no changes to application logic. It only requires changes to model partitioning. In contrast, deploying distributed applications using middleware and database servers requires the incorporation of distribution functionality in the applications themselves. In other words, middleware and database server support for distributed applications is little more than transporting data from one machine to another, which is the simple part of building distributed applications. ii) Throughput The Kabira Infrastructure Switch provides high application throughput by using an asynchronous model for all work.

Synchronous behavior is available to the application developer if required. The Kabira Infrastructure Switch uses an asynchronous processing model to minimize synchronization points during execution and to minimize the

degradation of system throughput they cause. As the Infrastructure Switch receives work from an external source, it queues the work until it can be processed. This allows Kabira’s Infrastructure Switch to absorb large traffic peaks from external sources without slowing down the source of the external work and without requiring programmers to write special code to handle these peak loads. This stands in sharp contrast with the typical command/response protocols used by database and existing ORB implementations, where additional work cannot be absorbed by a system until the current work is processed.

In those environments, application developers must implement complex schemes in their applications to achieve high throughput of work from external sources, such as multiple connections, processes, application-specific queuing, and so on. iii) Latency The Kabira Infrastructure Switch is a high-throughput, low-latency platform. Low latency is achieved through: Data caching Early binding Thread architecture Data Caching The Kabira Infrastructure Switch caches application data in shared memory. This puts application data as close to application processing as possible and eliminates the network traffic associated with data access in traditional client/server and middleware applications. In addition, the transaction-switch’s management of cached data is transparent to Kabira applications. Specifically, stale cache data is transparently refreshed from backing persistent stores and “ dirty data”, or data that has changed in the cache, is transparently written to backup persistent stores.

Application designers specify caching requirements when building an application model. Caching is specified in terms of the application objects

that should be cached and a valid lifetime for the cached data. The more application data in cache, the lower the application latency. Early Binding “ Early binding” is a term used in object-oriented literature to describe communication between objects that is determined before execution, or when applications are compiled. “ Late binding” is used to describe communication between objects that is not determined until runtime.

C++ is an example of a language that implements an early binding model, while Java is an example of a language and execution environment that implements late binding. Early binding trades flexibility for performance, while late binding provides much greater flexibility and support for application evolution at the cost of performance. The Kabira Infrastructure Switch employs both early binding and late binding. It uses early binding for application logic because application logic is implemented as compiled C++ code. It uses late binding when adding new functionality to a running system through a dynamic subscription process that does not disrupt applications that are already running. The combination of compiled C++ code and dynamic subscription provides the low latency and performance of a compiled language without giving up the ability to modify a running system.

Thread Architecture The Kabira Infrastructure Switch minimizes processing latency by eliminating thread context switching whenever possible. When a thread is processing work, it continues to process until it needs to wait for an external event. There are no special-purpose thread pools in the transaction-switch that can only perform specific tasks, and that increase the number of context switches. In addition, there are no “ polling loops” in the execution environment that spin while waiting for work.

Threads are blocked only to wait for external events. They process these events as soon as they are scheduled by the operating system. The Kabira Infrastructure Switch does not add any latency to detect that work has arrived. iv) Predictability Designers can determine application performance characteristics before deployment using a combination of application models and load estimates. Kabira application models provide a complete representation of the processing that will be performed by an application. In addition, the performance characteristics of the Kabira Infrastructure Switch are known, including the impact of object operations on processing, memory, and disk.

With these two things, the model and operation costs, designers can predict the performance of a production system. B) Robustness Robustness is a measure of a deployed application’s resistance to failure, its ability to recover following a failure, and its manageability. Robustness is also called high availability. There are many factors that affect the robustness of a deployed system.

These include: Application Recoverability System design Pre-release testing Fault management Maintenance The Kabira product family was designed and implemented to provide the robustness required to exceed the availability requirements of the most demanding environments. i) Application Recoverability Building recoverable applications is very difficult, especially in a distributed environment. Existing client/server and middleware platforms provide various levels of support for recoverability such as distributed transactions, non-distributed transactions, and reliable messaging. These very low-level services only ensure that a database update completes or a

message is delivered to another node. They do not directly provide support for failure recovery. Rather, the logic to recover or restart an application following a failure must be built into the application program.

For simple client/server applications that are putting data into a single database, recovery is not a large problem as the user just restarts the client. However, for complex distributed applications that consist of multiple data sources and multiple complex steps, recovery processing is very complex. Requiring an application designer to build this recovery logic for each application is expensive and time consuming. It also makes application changes after deployment difficult, as the application logic will be tightly integrated with recovery processing. The Kabira Infrastructure Switch provides transparent recovery support for applications built on the Switch.

These services recover not only the data associated with an application, but also the current processing state, allowing applications to restart from the last successful processing step. The Switch supports recovery for adapters that are both transactional and non-transactional. Recovery support for non-transactional adapters allows reliable delivery mechanisms to be built for adapters that were never intended to be reliable or recoverable. This provides a mechanism for integrating legacy data and processing sources into a recoverable application model. Recovery logic is not encoded in the application models that are deployed on Kabira’s runtime transaction-switch. This reduces the complexity of the application modeling.

While developers must handle application errors such as data integrity problems, they do not need to handle system resource errors such as

hardware, network, or external server failures. For example, if a Kabira application is using an external database and the database fails, the transaction-switch continues to retry the database connection until the database has been restarted. This retry logic is provided by the transaction-switch, not encoded in the application. Transactions All application processing in the Infrastructure Switch is performed in the context of a transaction. Each transaction contains a log in which before and after images of all changed data and all processed events are recorded.

When a transaction commits, the data changes are made permanent and the processed events are discarded. If a transaction aborts because of system failure, the data is restored to its original state, the application is restarted at the last known good state, and the events are replayed. Kabira transactions are mapped through to external databases. This ensures that data cached by the application in the transaction-switch is backed up and that both sets of data (cached and backup) are consistent with each other. The Kabira Infrastructure Switch uses transactions to ensure recoverability of applications, and to ensure that a consistent state is maintained with external adapters. Transaction management is handled by the runtime transaction-switch and is transparent to application designers.

This includes mapping application transactions to those on a backend database and propagating transactions in distributed applications across Kabira Infrastructure Switch nodes for reliable queuing of work. Reliable Queuing Communication between instances of the Kabira Infrastructure Switch on different nodes is done with asynchronous transactions via reliable queues. For queuing, a local transaction is started on the sending Kabira

Infrastructure Switch node and is terminated when the work has been successfully passed to the next node. Then the receiving node starts a new transaction in which to process this work. This transaction is independent of the sending node’s transaction. In other words, the reliable queuing mechanism used by Kabira’s transaction-switch technology uses distributed asynchronous transactions.

Distributed asynchronous transactions have advantages over distributed synchronous transactions (commonly called two-phase commit) in that they are more resilient to failure. In a distributed synchronous transaction any single point of failure in any resource participating in the transaction causes the entire transaction to be aborted. In addition once a required resource in a distributed synchronous transaction fails, the system cannot accept any additional work until all resources are restarted. These limitations severely restrict the usefulness of synchronous transaction models for mission-critical systems that must be highly available and responsive.

Distributed asynchronous transactions also perform better by avoiding global locking of resources. Distributed global locks do not scale well across multiple systems and they perform very poorly over wide-area networks, such as the Internet. ii) System Design Design highlights contributing to the Kabira Infrastructure Switch robustness include: No single point of software failure Complete transactional recovery (discussed in the Transactions section above) Automatic restart following a failure Process rejuvenation Resilient to external system failures such as databases or communications lines No Single Point Of FailureThere is no single point of software failure in the Kabira Infrastructure Switch. This is a minimal requirement for high

availability systems. The Kabira Switch achieves this using a multi-process model with all application data stored in shared memory. A single process failure does not affect functionality running in other processes.

Automatic Restart The Kabira Infrastructure Switch includes a system coordinator that restarts failed system components. The coordinator also starts and stops components of the Kabira Switch. There is no application logic running in the coordinator. When the system coordinator detects a component failure it aborts all transactions that are currently associated with that component to ensure that the system is restored to a known good state. It then restarts the process and the work continues. Process Rejuvenation Process rejuvenation is a mechanism whereby operating system processes are periodically restarted to reclaim accumulated resource loss and fragmentation (such as memory, files, etc.

) that occurs when processes are running for long periods of time. This functionality is required for systems that are designed never to be shut down. The Kabira Infrastructure Switch supports process rejuvenation by its ability to shut down any system component and restart it from a known state. Operators can restart processes at any time to reclaim lost resources.

This functionality makes it possible to integrate adapters into the transaction-switch that were never designed to run 7X24 since they can be periodically shut down and reinitialized. Resilient to External Failures The Kabira Infrastructure Switch is resilient to external failures because of its transparent retry of failed external resources and its distributed asynchronous transactions. Retrying work queued because of failed external

resources eliminates the requirement for a specialized application designed to handle external system failures. This simplifies application modeling since the Kabira Infrastructure Switch ensures that the queued work will eventually reach the external system. Handling external resource errors in application-specific code is difficult and can result in an unreliable system. The reliable queuing mechanism in the Kabira Switch ensures that work in a distributed environment will reach its destination once and only once.

Application designers do not need to worry about duplicate messages arriving for applications deployed on the Kabira Infrastructure Switch. This again simplifies application models and improves the robustness of deployed systems. iii) Pre-release testing A system that supports high availability is only successful if it is properly tested before release and sale to the market. Kabira’s engineering team built the Kabira Infrastructure Switch with testability in mind, allowing Kabira’s Quality Assurance team to validate both positive and negative execution paths.

The testability built into the transaction-switch allows test programs to request failures in any switch component during test execution. This includes the ability to cause any operating system API to fail. The Kabira transaction-switch’s formal testing methodology and implementation allows Kabira’s Quality Assurance organization to achieve measured code coverage approaching 100% for all released code. This achievement is currently unheard of elsewhere in the software industry. Kabira has a patent on the testing methodology used to achieve its high level of code coverage. Using code generators to build applications on top of Kabira Infrastructure Switch also ensures consistent use of the platfrom’s APIs.

This provides predictable runtime execution patterns and also eliminates standard programming errors such as uninitialized variables and memory leaks. The combination of thorough pre-release testing and application code generation allows Kabira to achieve very high levels of stability and correctness for both Kabira-provided components and applications built with the provided tools. This is even true in cases of rapid application changes. v) Fault Management Fault management is the ability to detect and correct system failures and congestion. The Kabira Infrastructure Switch provides complete monitoring support for all runtime processes and access to both switch and application state stored in shared memory. In addition, a Java client is provided to support remote switch and application management.

Providing a simple mechanism to build application management functionality ensures that management is built into applications as part of the initial design, not as an afterthought. Providing the operational tools required to monitor applications ensures that deployed applications can be supported by operations personnel using existing management tools. The Kabira Infrastructure Switch allows application designers to define not only MIB’s, but also the behavior associated with the managed objects. Code generators are included to completely generate the managed objects. No user-written code is required.

MIB specifications in SMI are generated to integrate the application-defined managed objects with existing system management stations. This is very different from the few tools that are available today that support automatic code generation from MIB definitions for SNMP. These tools do not generate the behavior of the managed objects, instead, they require developers to

hand-write it following code generation. v) Maintenance The Kabira Infrastructure Switch does not require periodic downtime for maintenance. The switch supports on-line upgrades for existing and new functionality.

New versions of adapters and application models can be released without impacting the entire system. Only the component being replaced must be restarted. All other components in the system continue to queue work, while the component being replaced is restarted by the Kabira System Coordinator and brought on-line. In addition, new functionality can be added to a running system. The Kabira Infrastructure Switch dynamically discovers the new functionality.

This allows systems to be changed without impacting existing functionality. The Switch never has to be completely shut down. This is a big improvement over existing middleware and database systems that must be shut down for bug fixes and deploying new applications. C) Development ProductivityJust providing runtime technology is not enough.

The infrastructure environment must also provide a significant increase in developer productivity, whether it is in new development, integration, or changing either of these down the line. The switch’s tools enable a significant increase in developer productivity while providing the performance and robustness already discussed. This is accomplished by: Separating business models from implementation Providing reusable adapters for existing applications and evolving standards i) Separating Business Models from ImplementationObject modeling and code generation allow developers to concentrate on solving business problems, rather than

on programming details. Directly generating an application from a model ensures that no details are lost between the business specification and actual implementation. .

It also means that the business models and implementation can be changed independently and rapidly. Business models are insulated from changes in underlying technology, and combined with the adapter technology described in the next section, the implementation can remain highperformant, recoverable, robust, and manageable, even as the models are changing. Kabira supports the industry-standard OMG Unified Modeling Language, and the newly adopted standard extension to it for coding actions, the UML Action Semantics. These ensure that models of a user’s system will apply across technologies and will be interchangeable across modeling tools.

The Kabira modeling environment supports auditing and fast generation of application models. Auditing validates that the model is complete. Fast generation of the model allows it to be executed in the development environment very easily. Using both together, the developer can test business requirements before deployment, through an iterative development cycle. Current middleware and database servers can achieve some of these goals, but at the expense of long, complex, and costly development cycles.

In contrast, the Kabira Infrastructure Switch allows mission critical applications to be rapidly implemented and changed, while maintaining the required performance, recoverability, robustness, and manageability. The Kabira Switch is designed to meet the tough operational needs of mission critical applications, and it is designed to be easy to change through the use

of model-based development and automated code generation. By enabling a business to upgrade a running system without having to shut the entire system down, Kabira Infrastructure Switch provides the flexibility to change and migrate deployed applications to meet new business requirements. A business can also install new functionality and maintenance releases without impacting current application processing. ii) Reusable Adapters Existing applications are a major investment and must be leveraged in new applications.

Kabira provides the means to wrap existing legacy systems and make them available to new applications. This is accomplished by modeling an adapter to the legacy system as an object, and generating code to provide C++ wrappers in which the API to the legacy system is encapsulated. Many of these adapters are provided prebuilt by Kabira, such as for Java and databases. Once this is done, the systems appear as objects in the modeling tool that can be used by application designers as required to build new applications. All mappings and interactions between adapters are done by the application modeler using the object-modeling tool.

No mappings are hard-coded in the adapter code. Experience has shown that 80% of the wrapper code can be generated from the model. Once an adapter is integrated it can be used by many different applications. It is easy to develop mappings between very diverse data sources such as mainframes and the Internet without hand coding. Any changes to the mappings are made in the model, and then code is regenerated.

This process is a very simple and straightforward way of managing and maintaining mappings and interactions between very different systems. Integrating legacy applications into the Kabira Infrastructure Switch has the additional advantage of providing a migration mechanism. Instead of completely replacing legacy applications, a new application can be designed to migrate data gradually from the legacy application. The process of moving the data from the old to the new application is captured in the application model.

Developers can also use a legacy adapter without having to understand the entire adapter in detail. These details are hidden when the adapter is integrated into the Kabira Switch. A developer who is not an expert in all of the adapters can still build an application. Ease of integration also means that applications built to today's " standards" can easily migrate to future standards as they are developed and accepted, thereby avoiding a " legacy trap". This includes the ability to run the same application on different standards such as CORBA, Java, or Web. For example, the use of web browsers as universal clients made fat client systems obsolete, which were the standard of the late 1980s.

In fact, new standard technologies can be integrated with the application without having to stop a running application. This provides a simple migration path to new standards as they become accepted without discarding current implementations. There is no legacy trap caused by rapidly changing technology. Existing middleware and database servers provide very limited out-of-the box integration with current systems and emerging standards.

Implementing new applications with these technologies usually requires a complete migration of the application or the development of a special-purpose gateway to access the legacy system. IV) ConclusionThe Kabira Infrastructure Switch and Product Suite has been designed expressly for today’s world where fast delivery of high performance, high availability, highly flexible Next Generation Services is essential to competitive advantage. Unlike traditional middleware, database, and application development products, the Kabira product family designed from the start with a complete set of requirements spanning multiple disciplines. Consequently, the Kabira software allows companies to leverage the systems they already have, mediate between networks operating at different speeds, and add new, high performance capabilities quickly and seamlessly