Security for Distributed E-Service Composition

Stefan Seltzsam, Stephan Börzsönyi, and Alfons Kemper

Universität Passau
Fakultät für Mathematik und Informatik
D-94030 Passau, Germany

$\langle$*seltzsam*|*boerzsoe*|*kemper*$\rangle$`@db.fmi.uni-passau.de`
`http://www.db.fmi.uni-passau.de/`

Abstract. Current developments show that tomorrow's information systems and applications will no longer be based on monolithic architectures that encompass all the functionality. Rather, the emerging need for distribution and quick adaptation to new requirements stemming from, e.g., virtual enterprises, demands distributed systems that can be extended dynamically to compose new services from existing software components. However, usage of mobile code introduces specific security concerns which a security system must be aware of. We present a comprehensive security architecture for extensible, distributed systems using the example of an Internet query processing service which can be extended by user-defined operators. Before an operator is actually used in queries for the first time, our `OperatorCheck` server validates its semantics and analyzes its quality. This is done semi-automatically using an oracle-based approach to compare a formal specification of an operator against its implementation. Further security measures are integrated into the query processing engine: during plan distribution secure communication channels are established, authentication and authorization are performed, and overload situations are avoided by admission control. During plan execution operators are guarded using Java's security model to prevent unauthorized resource access and leakage of data. The resource consumption of operators is monitored and limited with reasonable supplementary costs to avoid resource monopolization. We show that the presented security system is capable of executing arbitrary operators without risks for the executing host and the privacy and integrity of data. In the paper we will concentrate on the `OperatorCheck` server, as this server can itself be viewed as an e-service that can be used by developers and independent associations.

1 Introduction

The recent trend towards extensible and distributed systems demands new sophisticated security systems to meet the challenges of mobile, user-defined code. Examples for such systems are Web browsers executing Applets, Web application servers executing Servlets and Java Server Pages, and extensible database management systems implementing, e.g., the SQL99 [22] standard for user-defined functions. Nowadays even more and more wireless devices execute code in form of WML scripts [9], e.g., to interact with a mobile e-commerce system. In this paper we use ObjectGlobe [2], a distributed and extensible query processor written in

Java, as an example for such a system. To describe it in short, ObjectGlobe can execute a query with—in principle—unrelated query operators, cycle providers (used to execute operators), and data sources.

The openness of a system like ObjectGlobe creates new demands on a security system. The code of external, i.e. user-defined, operators is unknown. Thus, users of operators do not know, if the operator calculates the correct result, crashes, or manipulates data given to it. For this reason quality assurance is necessary, because users of external operators want to feel certain about the semantics and functioning of operators to be able to rely upon the results of a query. As in every distributed system, it is necessary to protect communication channels against tampering and eavesdropping. Cycle providers execute arbitrary external operators, thus cycle providers need a security architecture which prevents external operators from accessing resources like the file system of the cycle provider, monopolizing resources like memory or CPU time, or manipulating vital components of the ObjectGlobe system. Additionally, cycle providers need an authorization framework to be able to determine the identity of a user.

The goal of this work is to provide a security architecture for ObjectGlobe addressing all mentioned challenges, which is generally appropriate for distributed and extensible systems. Due to space limitations we will concentrate on quality assurance. In order to achieve these goals we have to rely upon some basic assumptions about the environment of ObjectGlobe. First, we assume that the operating system which is running ObjectGlobe is secure and that the administrators of the cycle provider are trustworthy, because we cannot protect the system against the operating system. Second, we assume that the code and the Java Virtual Machine (JVM) used to run ObjectGlobe are unmodified. These requirements are enforced in ObjectGlobe by giving the user the possibility to restrict the cycle providers to a set of trusted cycle providers. The last and most serious assumption is, that the security system of Java 2 [19] works as designed. There have been some security related bugs and implementation flaws of Java, but it seems that there are no elementary flaws in the design of the security model.

The remainder of this paper is organized as follows: Section 2 gives an overview of ObjectGlobe. Section 3 points out the security requirements of such an open system. Section 4 gives a survey about the security measures during plan distribution and Section 5 outlines runtime guarding and monitoring measures used to detect malicious or defective operators. Section 6 describes in detail preventive measures appropriate to detect the majority of low quality and malicious external operators before actually executing them in queries. Related work is addressed in Section 7 and Section 8 concludes this paper.

2 Overview of the ObjectGlobe System

ObjectGlobe distributes powerful query processing capabilities (including those found in traditional database systems) across the Internet. The idea is to create an open market place for three kinds of suppliers: *data providers* supply data, *function providers* offer query operators to process the data, and *cycle providers* are contracted to execute query operators. Of course, a single site (even a single machine) can comprise all three services, i.e., act as data-, function-, and cycle provider. In fact, we expect that most data and function providers will also act as cycle providers. ObjectGlobe enables applications to execute complex queries which involve the execution of operators from multiple function providers at different sites (cycle providers) and the retrieval of data and documents from multiple data sources. In this paper we assume that all data is in a standard format (e.g., relational or XML) or wrapped [21]. Furthermore, we assume that there is a meta-schema that can be used to describe all relevant properties of all services.

2.1 Query Processing

Processing a query in ObjectGlobe involves four major steps (Fig. 1):

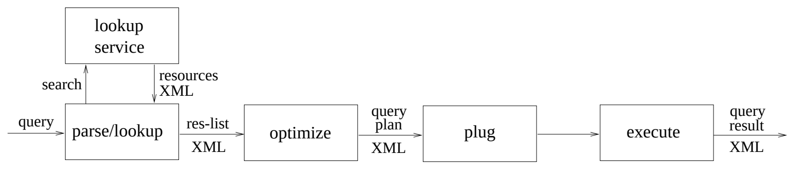


Fig. 1. Processing a Query in ObjectGlobe

1. Lookup: In this phase, the ObjectGlobe lookup service [16] is queried to find relevant data sources, cycle providers, and query operators that might be useful to execute the query. In addition, the lookup service provides the authorization data—mirrored and integrated from the individual providers—to determine what resources may be accessed by the user who initiates the query and what other restrictions apply for processing the query.
2. Optimize: The information obtained from the lookup service is used by a query optimizer to compile a valid (as far as user privileges are concerned) query execution plan. This plan—represented as XML document—is annotated with site information indicating on which cycle provider each operator is executed and from which function provider the external query operators involved in the plan are loaded. Such annotations are used as well for other information, e.g., authentication data and estimated resource requirements.
3. Plug: The generated plan is distributed to the cycle providers and external query operators are loaded and instantiated at each cycle provider. Furthermore, the communication paths (i.e., sockets) are established. If necessary, communication is encrypted and authenticated.
4. Execute: The plan is executed following an iterator model [11]. In addition to the external query operators provided by function providers, ObjectGlobe has built-in query operators for selection, projection, join, union, nesting, unnesting, and sending and receiving data. The execution of the plan is monitored in order to detect failures.

The whole system is written in Java for two reasons[1]. First, Java is portable so that ObjectGlobe can be installed with very little effort; in particular, cycle providers which need to install the ObjectGlobe core functionality can very

[1] Currently, the optimizer is written in C++, but we plan to rewrite it in Java.

easily join an ObjectGlobe system. The only requirement is that a site runs the ObjectGlobe server on a JVM. Second, Java provides secure extensibility. Although many people complain about the execution speed of Java programs, we noticed that by avoiding some pitfalls in the Java I/O library the execution speed of the Java Virtual Machine is no bottleneck in wide area distributed systems. Like ObjectGlobe itself, external query operators are written in Java: they are loaded on demand (from function providers). Just like data and cycle providers, function providers and their external query operators must be registered in the lookup service before they can be used. Up to now ObjectGlobe uses a simple classification system for operators: every external operator is associated with a class (e.g., Join). The optimizer chooses the operator actually used within a class considering different aspects, e.g., execution time and time needed to load the operator.

2.2 Example Plan

Figure 2 shows an example of distributed query processing with ObjectGlobe. Suppose you are going on holiday to Nassau, Bahamas, and you are looking for a hotel that is cheap and close to the beach. This task is known as the "maximum vector problem" [20] (actually we are searching for the minimal vectors). Formulated precisely, we are looking for all hotels, which are not dominated by other ones. A hotel dominates another one, if it is cheaper *and* closer to the beach. Dominance imposes a partial ordering on the hotels.

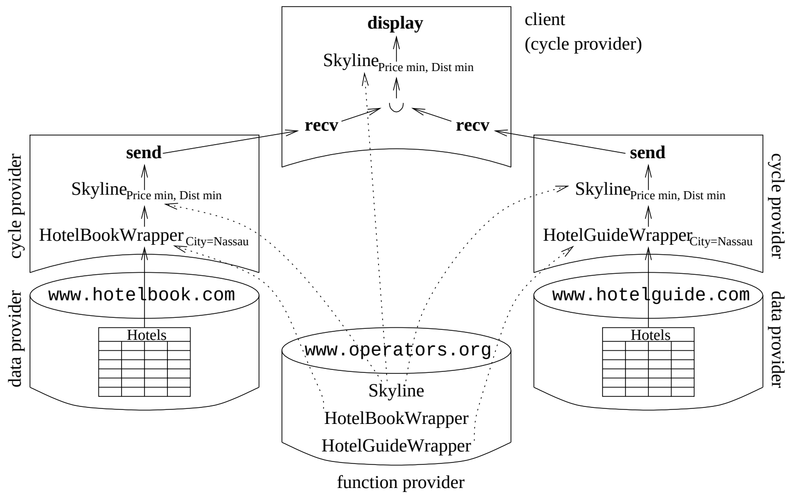


Fig. 2. Processing a Query in ObjectGlobe

A naive solution for this problem is to compare each hotel against each other and delete dominated hotels yielding quadratic runtime. However, a more sophisticated algorithm with a lower complexity has been developed by [17]. [1] have investigated this algorithm in the context of databases and adapted it to constrained primary memory. They called the operation "Skyline".

The resources used to find the desired cheap hotel near the beach are as follows: two web sites, `www.hotelbook.com` and `www.hotelguide.com`, supply hotel data and all external operators are provided by the function provider `www.operators.org`. Two wrappers, HotelBookWrapper and HotelGuideWrapper, are responsible for querying the two web sites and transforming the data into ObjectGlobe's internal format. They are executed at cycle providers located near the data sources in order to minimize transfer time. As the following equation holds for the Skyline operator, it can be applied to each database directly in order to further reduce data shipping costs:

$$Skyline(Skyline(A) \cup Skyline(B)) = Skyline(A \cup B)$$

Thus, only the best hotels are passed to the client. The send/receive iterator pairs performing the transmission of data are installed automatically during the plug phase. The client calculates the Skyline of the union of both data sources, and the user can choose a hotel from the result.

3 Security Requirements

We will now analyze the security requirements of users and cycle providers. First, we will concentrate on security issues introduced by external operators: cycle providers want to be able to execute arbitrary external operators in a safe way, i.e., they want to be sure that operators do not monopolize resources, access resources like the file system unauthorized or manipulate vital components of the system. We now demonstrate some attacks and the implications they would have without an effective security system using modified versions of the Skyline operator.

Example *Resource Monopolization*

```
public class Skyline extends IteratorClass {
  public TypeSpec open() throws CommandFailedException,IOException {
    List l = new LinkedList();
    while(true)
      l.add(new Object());
    ...}...}
```

The example above shows a code snippet which monopolizes the memory by continuously generating and storing new `Object` instances in a `LinkedList`. The execution of this operator would result in a denial of service because there would be not enough memory for other operators. Of course, an external operator could just as well monopolize CPU time, secondary memory, or any other limited shared resource. It would also be possible for an operator to, e.g., access the file system to modify arbitrary files or to steal confidential data.

Apart from security requirements of providers there are requirements of users. They want to be certain about the semantics of external operators to be able to rely upon the results of a query even when they use external operators. Recall the query using a Skyline operator to find the hotels which are cheap and near the beach. And now assume a modified Skyline operator that filters all hotels of the Sheraton hotel chain. The result of a query using this modified Skyline would be all cheap hotels near the beach being not member of the Sheraton hotel chain, which is not the result the user asked for. Privacy is another severe requirement endangered by external operators: an operator could calculate the result as supposed to do, but it could send a copy of the processed data to an arbitrary host in the Internet or leak data to a concurrent query, possibly compromising confidentiality of data. For that reason it is necessary that the security system can guarantee, that data given to an operator can only flow using communication channels which are obvious to and authorized by the user.

In addition to the security requirements induced by external query operators there are some which are common to many distributed systems. First, using our terminology, cycle and data providers may have a legitimate interest in obtaining the identity of users for authorization purposes. It must be considered that users normally want to stay anonymous as far as possible, therefore it must not be mandatory to give authentication data to cycle providers. Second, the communication channels between different collaborating hosts must be protected against tampering to avoid unnoticed modifications of the data. Additionally it must be possible to encrypt confidential data to prevent other parties from eavesdropping. Third, cycle providers need an admission control system to guard themselves against overload situations.

To meet these requirements we use a multilevel security architecture combining preventive measures, security measures during plan distribution, and a runtime security system. Preventive measures take place before an operator is actually used in queries for the first time. They are used to validate the semantics and analyze the quality of the operator. Based upon the validation results, ObjectGlobe could renounce runtime security measures. As preventive measures are optional, untested operators are regarded as malicious and all security measures apply. During plan distribution, common security measures of distributed systems take place including admission control. The remaining security requirements, e.g., protection of cycle providers, are met by the runtime architecture. For length limitations we give a brief overview of the mandatory security levels, i.e., security measures during plan distribution and the runtime security architecture, in the next two sections. Thereafter we present the preventive measures in detail and point out the advantages of validated operators.

4 Security Measures during Plan Distribution

We will now have a look at the common security requirements of distributed systems, as enumerated above. We meet these requirements during plan distribution, where four security related actions take place: setup of secure communication channels, authentication, authorization, and admission control.

Privacy and integrity of data and function code that is transmitted between ObjectGlobe servers is protected against unauthorized access and manipulation by using the well-established secure communication standards SSL (Secure Sockets Layer) and/or TLS (Transport Layer Security) for encrypting and authenticating (digitally signing) messages. Both protocols can carry out the authentication of ObjectGlobe communication partners via X.509 certificates [14], thus ensuring communication with the desired ObjectGlobe server. The security level of network connections can be chosen dependent on the processed data.

If authentication is required for authorization or accounting purposes of providers, ObjectGlobe can authenticate users using one of two possibilities: digitally signed plans or password-based authentication. Of course, usage of X.509 certificates is preferred, but until certificates are widely used, password-based authentication is supported as an alternative. Certificate based authentication requires plans with signatures generated using the secret key of the user. The signature is based on the XML document containing the query plan and a time stamp to avoid reusage of signed plans. The signature of a query arriving at a provider is verified using the user's X.509 certificate and thereafter the originator and the integrity of the query is known reliably. Providers can use this knowledge to enforce their local authorization policy autonomously. Of course, users and applications accessing only free and publicly available resources can stay anonymous and no authentication is required. If a user wants to access a resource that charges and accepts electronic payment, the user can remain anonymous as well (if the electronic payment system supports it) and the electronic payment is shipped as part of the plug phase.

The last security related action during plan distribution, which is needed by all systems offering services to users, is admission control. ObjectGlobe's admission control component determines whether the estimated resource demands of a query—calculated using the cost models of the operators—can be satisfied by the executing host or not. This is done to be able to abort a query as early as possible if any cycle provider executing a part of the query cannot satisfy the resource requirements of the query. Queries whose resource requirements can be fulfilled are scheduled using a FCFS (first come first served) scheduler considering only primary memory usage of the operators. We assume, that a sufficient amount of all other resources like secondary memory is available. This scheduling approach works well, because every query is only allowed to use a small fraction of the primary memory of the server, so a certain degree of parallel execution is guaranteed.

5 Architecture of the Runtime Security System

After plan distribution, all involved cycle providers execute the operators assigned to them to calculate the result of the query. Therefore, they must be protected from damages by malicious or low quality operators as outlined above. To satisfy security interests of users, the security system must also be able to guarantee, that data given to an operator can only flow using communication channels which are obvious to and authorized by the user. These security requirements are met using two techniques: guarding and monitoring.

The guarding mechanisms are realized using Java's security architecture, i.e. security manager and class loaders—to control and restrict access to resources of the cycle provider and components of the ObjectGlobe system. The class loader's task is to load the bytecode of a class into memory, monitor the loaded code's origin (i.e., its URL), and to verify the signatures of signed code. Additionally, every class loader generates its own name space. To prevent external operators from abusing a connection to a function provider as covert communication channel by requesting classes with data coded into the names of the classes, all (non built-in) classes required by the external operator must be combined into a JAR file. This file is loaded and cached by the class loader during the plug phase, so only one connection to the function provider is necessary. The security manager controls the access to safety critical system resources, such as the file system, network sockets, peripherals, etc. The security manager is used to create a so-called *sandbox* in which untrusted code is executed. Privileges can be granted to code based on the origin and whether or not it is digitally signed. Of course, it would be unreasonable to grant unprotected access to system resources to unknown code. Therefore, all user-defined operators are normally executed in a "tight" sandbox. Furthermore, queries running in parallel are separated from each other to prevent them from exchanging information with each other via, e.g., static class variables. This is done by using a new class loader instance (called `OGClassLoader`) for each query which implicitly separates the name spaces. In this way, external operators are isolated and leakage of data is prevented, because they are only able to communicate with their children and parent operators. Additionally, only selected classes of the name space of the ObjectGlobe system are known to the name spaces of operators, thus protecting vital components of the system. For example, `TmpFile` is a class available to operators and implements a secure interface to the file system to enable operators to use temporary files. The sandbox, the name space separation, and the class loaders are illustrated schematically in Fig. 3.

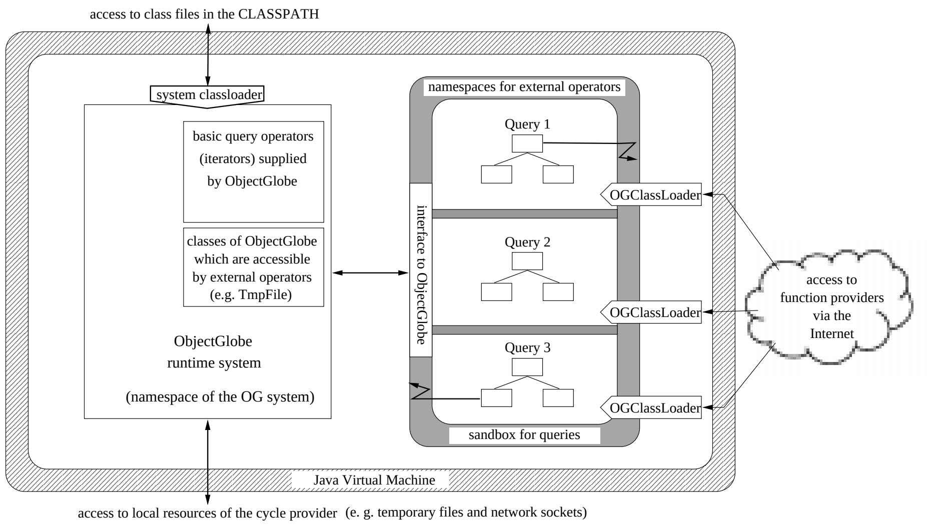

access to class files in the CLASSPATH
system classloader
basic query operators (iterators) supplied by ObjectGlobe
classes of ObjectGlobe which are accessible by external operators (e.g. TmpFile)
ObjectGlobe runtime system
(namespace of the OG system)
namespaces for external operators
Query 1
Query 2
Query 3
interface to ObjectGlobe
OGClassLoader
OGClassLoader
OGClassLoader
sandbox for queries
access to function providers via the Internet
Java Virtual Machine
access to local resources of the cycle provider (e. g. temporary files and network sockets)

Fig. 3. Protection of the Resources of Cycle Providers

Monitoring measures are necessary to avoid resource monopolization. We use our own (platform-dependent) resource accounting library which supervises CPU and primary memory usage of external operators[2], because Java does not offer such functionality. Accounting of secondary memory and data volume produced by an operator is done using pure Java. External operators must be endowed with (worst-case) cost models written in MathML [15] for their CPU usage, consumption of primary and secondary memory, number of temporary files in

[2] Using our library, accounting results in overheads between 5% and 10%.

use simultaneously and the number and size of tuples they produce. The last two cost models are necessary to prevent operators from blocking the network and from flooding their parent operators. If an external operator is not equipped with its own cost models, ObjectGlobe uses default cost models, which of course might not be very appropriate. The optimizer annotates query plans with estimated values for the number and size of the tuples that the different operators will produce. Using these values and the cost models a cycle provider is able to calculate the initial resource limits for an external operator. To be able to keep track of resource consumption, every external operator is executed by a separate thread and disconnected from other operators using buffers, each managed by a send/receive iterator pair.

If there is any limit violation, the monitoring component checks, if the limits have to be adapted dynamically. This is done, if changes to the calculation basis of the cost models occurred, e.g., the optimizer estimated that the underlying operator will produce 100 tuples but it produced 110 tuples up to now. If the newly calculated resource demands of the operator exceed the upper resource limits set by the cycle provider, the plan is aborted. The plan is also aborted, when the newly calculated resource limits are still too low to satisfy the current resource demand of the operator. Otherwise the operator is allowed to resume work until there is another limit violation or the operator terminates normally.

6 Quality Assurance for External Operators

Using the security measures presented up to now, we are able to execute arbitrary external operators without risk for cycle providers and privacy and integrity of data. Nevertheless, it would be advantageous, if the system could in advance verify the semantics of new external operators, examine their behavior under heavy load, and compare their resource consumption with given cost models. If an operator is well-behaved, ObjectGlobe could renounce security measures and execute the code at full speed or it could relax the sandbox of an operator. Several methods of software verification and testing have been developed so far (see [18] for an overview), but it has also been shown that in general the correctness of arbitrary code cannot be proved [13].

6.1 The Goal of Testing

Testing is a verification technique used to find bugs by executing a program. The testing process consists of designing test cases, preparing test data, running the program with these test data, and comparing the results to the correct results. An oracle[3] is consulted for the correct result. This could be a human, a reference implementation, or an interpreter of a (formal) specification of the program. While the design of good test cases requires some ingenuity, test data can sometimes be derived automatically. For automated testing, a test driver is necessary to feed test data to the function and to receive and record the results.

Testing methods, actually methods for deriving test cases, can be divided into two classes, white-box and black-box testing, depending on whether the source code is available or not. [18] provides a detailed description of the most important techniques. We are focusing on black-box testing.

[3] We think of an oracle in the true sense of the word, not of the commercial DBMS.

6.2 Methods of Formal Specification

As the correctness of a program depends on what exactly it is supposed to do, a complete and consistent specification is necessary. If testing should be processed automatically, a formal specification is required so that an interpreter can determine whether a calculated result is correct. There are two classes of formal specifications: *Operational techniques* describe a way how the result could be calculated. Their advantage is that the correct result can be determined in advance and compared to the result of the program. However, they will not choose the most efficient way and hence are not a viable alternative to the real program. In contrast to that, *descriptive techniques* specify what the result should look like. Although the correct result cannot be calculated, the result of the program can be checked against them. Moreover, they usually are even more concise than operational specifications.

We have investigated several methods of formal specifications, e.g., SQL, Haskell, Prolog, and mathematical formulae. Table 1 shows these specification methods for the Skyline operator. For our purpose, the best choice is to use a (purely) functional language like Haskell, because coding is quite straightforward and functional languages are Turing-complete and thus (theoretically) every operator can be specified that way. Another advantage of Haskell is that there are interpreters that can execute the specification and thus the correct result can easily be calculated.

Especially in the database context, not only the correctness of the result is important but also the efficiency of its computation. Therefore, the specification of an external operator is augmented with several cost models. For each supervised resource the user can specify a worst-case cost model, which is a function of the extents of the input relations, namely their number of tuples, their maximum tuple size, and their total data size. If any resource is overconsumed, the operator is considered faulty and aborted immediately in a real application. Nevertheless, the cost models should not be chosen too generous, because then a cycle provider might refuse to instantiate the operator.

6.3 User-Directed Test Data Generation

As stated before, the design of good test cases requires some ingenuity. Thus, in our implementation, it is possible to direct the generation of test data so that they fulfill the preconditions of operators as well as meet the testers' strategies. Testers may want to specify single attribute values, enforce functional dependencies between attributes, establish relationships between relations, and control the order of the tuples. Therefore, the generation is done in three steps. First, the relation is created and the attribute types and the number of tuples are specified. Second, all attribute rows are filled by random values or by referencing other relations. Third, the relation can be sorted or permuted some other way.

For a single attribute column, the values can be generated randomly or deterministically. The latter means that the values are taken one after the other in increasing order. If there are more tuples than different values, the procedure is started cyclically again. Random values can be taken uniformly from their possible values. In order to simulate functional dependencies and primary keys,

Table 1. Specification Methods for Database Operators (Skyline)

	Skyline (S,$\succeq$)	Explanation
Formula	$\{s \mid s \in S \wedge \neg\exists t \in S : t \neq s \wedge t \succeq s\}$	This formula can be derived directly from the definition: "The Skyline of a set S consists of all tuples s that are in S and for which no tuple t exists in S that is different from s and dominates s."
Conditions	Pre $\equiv$ **true** Post $\equiv \forall s \in \text{Skyline}(S)$: $(s \in S \wedge \neg\exists t \in S : t \neq s \wedge t \succeq s)$ $\wedge\ \forall s \in S \setminus \text{Skyline}(S)$: $(\exists t \in S : t \neq s \wedge t \succeq s)$	There is no precondition, i.e., the Skyline operator can be applied to any arbitrary set on which a partial ordering relation is defined. The postcondition describes which tuples may be in the Skyline (cf. the formula above) and which must not be left out, defining exactly the result.

SQL	`SELECT *`
	`FROM S s`
	`WHERE NOT EXISTS (`
	`  SELECT * FROM S t`
	`  WHERE t≠s AND t⪰s);`
		This is the naive approach to calculate the Skyline. Each tuple is compared to each tuple and is only selected if it is not dominated by any other tuple. $\neq$ and $\succeq$ must be adapted to the specific scenario.

Prolog	`skyline(S,R) :- skyline'(S,S,R).`
	`skyline'([],T,[]).`
	`skyline'([X|S],T,R) :-`
	`  dominated(X,T),`
	`  skyline'(S,T,R).`
	`skyline'([X|S],T,[X|R]) :-`
	`  not(dominated(X,T)),`
	`  skyline'(S,T,R).`
	`dominated(X,[Y|T]) :-`
	`  dominance(Y,X).`
	`dominated(X,[Y|T]) :-`
	`  dominated(X,T).`
	`dominance(Y,X) :- Y≠X, Y⪰X.`
		The Skyline of a list S is R, if the result of a function `skyline'` that filters S with itself, i.e., compares each tuple with each tuple and deletes dominated tuples, is also R. If the empty list `[]` is filtered, the result is also empty.
		Now consider a list that contains at least one element X. If X is dominated by any tuple of the filter T, the result consists only of the rest of the list still to be filtered by T. Otherwise, X is taken over into the result.
		X is dominated by a non-empty list, if it is dominated by the first element or by the rest. If the list is empty, X is considered not dominated (closed world assumption).

Haskell	`skyline :: [α] → [α]`
	`skyline ss = skyline' ss ss`
	`skyline' [] ts = []`
	`skyline' (s:ss) ts =`
	`  if dominated s ts`
	`     then skyline' ss ts`
	`     else s:skyline' ss ts`
	`dominated s [] = False`
	`dominated s (t:ts) =`
	`  dominance t s || dominated s ts`
	`dominance t s = (t≠s && t⪰s)`
		`skyline` is a function that takes a list of elements of some type α and returns a list of elements of the same type. Like in Prolog, the Skyline of a list ss is the result of a function `skyline'` that filters ss with itself.
		Again, there is a distinction between the empty list `[]` and a list containing at least one element. This element is only taken over into the result if it is not dominated by any element of the filter ts.

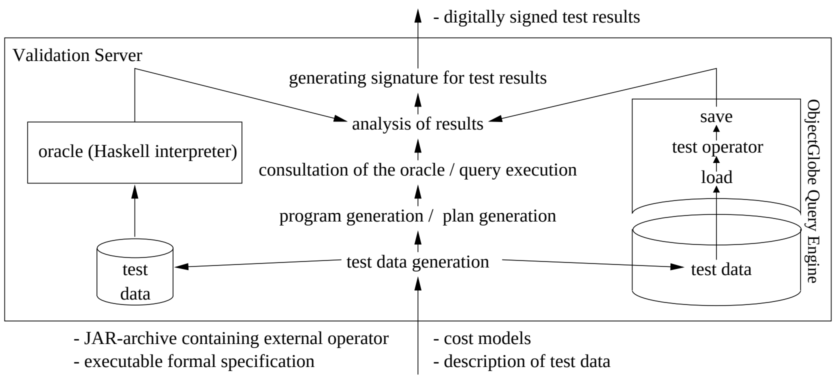
it is important that unique values can be generated. [7] presents an algorithm that produces random numbers with this property. For Real values, other distributions are possible, e.g., normal distribution, exponential distribution, etc. Foreign key relationships can also be simulated. For 1:1 relationships, the attributes of the other relation can be copied one after the other or be referenced unique-randomly. For 1:N relationships, a uniform random reference should be applied. Occasionally the order of the tuples matters. Thus, the relation can be sorted by the values of an attribute. Moreover, a shuffle operation has been implemented that permutes the tuples of a relation. This is useful to create a slight disorder. A factor between 0.0 (identity) and 1.0 (completely random shuffle) describes how far a tuple can move relative to the cardinality of the relation.

6.4 The OperatorCheck Server

We have implemented a server that checks external operators in ObjectGlobe by performing some tests on them. This server can itself be viewed as an e-service that can be used by developers during development to test the implementation of operators. Trustworthy independent associations can use the server to check ex-

ternal operators and to generate digitally signed test reports. For demonstration purpose, an installation is accessible via a Web interface at
`http://www.db.fmi.uni-passau.de/projects/OG/OnlineDemo/operatorcheck.phtml`

Figure 4 shows the architecture of the server. The tester provides a Java Archive containing the external operator to be tested, a formal specification of the operator in Haskell (for a correctness test) or a cost model (for a benchmark test), and some directives on how the test data should look like.



- digitally signed test results
Validation Server
generating signature for test results
analysis of results
oracle (Haskell interpreter)
consultation of the oracle / query execution
program generation / plan generation
test data generation
test data
save
test operator
load
test data
ObjectGlobe Query Engine
- JAR-archive containing external operator
- executable formal specification
- cost models
- description of test data

Fig. 4. Architecture of the Operator Check Server

For a *correctness test*, the server generates test data based on the directives and stores them on the hard disk. A Haskell program is built out of the formal specification and the generic ObjectGlobe query execution plan is assembled. Now the test is performed: The Haskell interpreter and the ObjectGlobe system calculate their results. Afterwards, the results are loaded and compared, and the user receives the result of the test. For comparison, the semantics of the result must be taken into account. If the resulting relation is a list, the order of the tuples and hence their count is important. A multiset is a set where an element may occur several times. The order of the elements, however, is arbitrary. In a set, neither order nor count of elements matters. Furthermore, it is possible to perform a *reference test*. Instead of providing a Haskell specification, the user can also nominate another ObjectGlobe operator that should serve as the oracle. The testing process works in an analogous way.

In a *benchmark test*, no oracle is consulted, but the test operator is executed several times using different sizes of input data. Instead of a formal specification, the user provides cost models for several resources. The consumption of these resources is measured and compared to the cost models. The test result shows the actually consumed resources and the maximum allowed by the cost models. Using large input sizes, a stress test can be carried out that examines the behavior of the operator under heavy load and checks whether its performance degenerates or is still in accordance with the cost models.

6.5 Limitations of Testing

E. W. Dijkstra noted that “program testing can be used to show the presence of bugs, but never to show their absence” [4]. Nevertheless, testing provides a practicable and promising way to find bugs in a piece of code, thus improving the trust in it. Several sophisticated methods of deriving "good" test cases have been developed. Under the hypothesis that the tested program behaves the same way for all test data of an equivalence class, the correctness can even be guaranteed by successful tests with one representative of each class. Malicious operators, however, intentionally destroy the uniformity hypothesis. This can only be detected by a white-box test which can inspect the source code. Therefore, it is still necessary to take some more measures for absolute security.

7 Related Work

There are a lot of extensible database systems allowing the implementation of user-defined functions as predicates or general functions/operators, in C, C++, or Java. Examples for such systems include POSTGRES [23], Iris [25], and Starburst [12], but there are also several commercially available systems like Informix, Oracle, and DB2. These systems are all more or less exposed to the same security risks as ObjectGlobe even if they do not load untrusted code dynamically from function providers like ObjectGlobe. The security measures of most systems are not appropriate to guard the database system against attacks from such code. Thus, only administrators are allowed to augment the functionality and they have to take care that the extensions are well-behaved and non-malicious. For example, DB2 implements the SQL99 standard for user-defined functions and provides the possibility to specify a function as `FENCED` to execute it in an own process. In this way, the internal structures of the database system are protected, but denial of service attacks are still possible. Only recently, with the development of Java as a secure programming language, some new considerations have been taken into account. [10] have compared the efficiency of several designs using the PREDATOR database server, namely the naive approach of putting user-defined functions directly into the server process, running them in a separate process and communicating with the server process via shared memory, and accessing Java user-defined functions via the Java Native Interface. Their conclusion is that Java is a bit slower in the average case but a viable and secure alternative, still facing the problem of denial of service attacks. [5] recommend to use a resource accounting system like JRes to guard against denial of service attacks, bill users, and obtain feedback that can be used for adaptive query optimization. To neutralize resource-unstable functions, they restrict CPU usage, number of threads, and memory usage to a fixed limit, which is not appropriate for complex operators.

Beside database systems, there are, e.g., Java operating systems, which have to care about security, because enforcing resource limits has been a responsibility of operating systems for a long time. Another task of operating systems is to separate applications to avoid interference. Using our security system, a Java operating system could limit the amount of damage that a compromised application could do to the system and other applications on the system as described in [6]. This paper proposes the usage of Trusted Linux as secure platform for e-services application hosting, because it adequately protects the host platform as well as other applications if an application is attacked and compromised.

The OperatorCheck approach is used to validate the semantics and to analyze the quality of operators. Thus, the quality of service of validated operators is higher than of untested operators. This leads to a more reliable query execution, continuously available cycle providers, and to better result quality. A more detailed motivation for the importance of these aspects can be found in [24]. Obviously, for upcoming e-service composition frameworks like eFlow [3] and E-speak [8] those quality considerations will also play a very important role.

8 Conclusion

We presented an effective security framework for distributed, extensible systems and used ObjectGlobe as an example. We focused on security requirements of cycle providers and users. The security requirements of users are satisfied by the OperatorCheck server which is used to rate the quality of external operators and test their semantics. Privacy and integrity of data are guaranteed by isolating external operators and by usage of secure communication channels. Cycle providers are protected using a monitoring component which tracks resource usage of external operators to prevent them from resource monopolization and an admission control system to guard providers against overload situations. A security manager and class loaders are used to protect cycle providers from unauthorized resource accesses and to shield the ObjectGlobe system from external operators. Additionally we presented the authorization framework of ObjectGlobe which can be used by cycle providers to determine the identity of users in a reliable way.

The security system can easily be adapted to other applications, e.g., Web application servers using server-side Java components such as Servlets, Java Server Pages and Enterprise Java Beans to generate dynamic Web content. Nowadays it is common, not to operate an own Web server, but to out source this to specialized suppliers. Using our security system, suppliers of such services can set resource limits to, e.g., Java Server Pages. Of course, there are some necessary adaptions to the security system. For example, server-side components usually do not have real cost models, but in most cases (e.g., primary and secondary memory) it is sufficient to use fixed resource limits. Additionally, the resource monitoring component can be used to establish a "per-resource" instead of, for example, a flat-rate tariff structure.

References

1. S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proc. IEEE Conf. on Data Engineering*, pages 421–430, Heidelberg, Germany, 2001.
2. R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous Query Processing on the Internet. *The VLDB Journal: Special Issue on E-Services*, 2001. To appear.
3. F. Casati, S. Ilnicki, L.-J. Jin, and M.-C. Shan. An Open, Flexible, and Configurable System for Service Composition. In *Proceedings of the Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, pages 125–132, Milpitas, California, 2000.
4. O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. *Structured Programming.* Academic Press, Inc., New York, 1972.

5. G. Czajkowski, T. Mayr, P. Seshadri, and T. v. Eicken. Resource Control for Database Extensions. Technical Report 98-1718, Department of Computer Science, Cornell University, November 1998.
6. C. Dalton and T. H. Choo. An Operating System Approach to Securing E-Services. *Communications of the ACM*, 44(2):58–64, February 2001.
7. D. DeWitt. The Wisconsin Benchmark: Past, Present, and Future. In J. Gray, editor, *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 2. edition, 1993.
8. S. Frolund, F. Pedone, J. Pruyne, and A. v. Moorsel. Building Dependable Internet Services with E-speak. Technical Report HPL-2000-78, Hewlett-Packard, 2000.
9. A. K. Ghosh and T. M. Swaminatha. Software Security and Privacy Risks in Mobile E-Commerce. *Communications of the ACM*, 44(2):51–57, February 2001.
10. M. Godfrey, T. Mayr, P. Seshadri, and T. v. Eicken. Secure and Portable Database Extensibility. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 390–401, Seattle, WA, USA, June 1998.
11. G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, June 1993.
12. L. M. Haas, W. Chang, G. M. Lohman, J. McPherson, P. F. Wilms, G. Lapis, B. Lindsay, H. Pirahesh, M. J. Carey, and E. Shekita. Starburst Mid-Flight: As the Dust Clears. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):143–160, March 1990.
13. J. Hartmanis and J. E. Hopcroft. Independence Results in Computer Science. In *SIGACT News*, volume 8, pages 13–24, 1976.
14. R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. `http://www.rfc-editor.org/rfc/rfc2459.txt`, January 1999.
15. P. Ion and R. Miner. Mathematical Markup Language. `http://www.w3.org/Math/`, July 1999.
16. M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. Verteilte Metadatenverwaltung für die Anfragebearbeitung auf Internet-Datenquellen. In *Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW)*, Informatik aktuell, pages 107–126, New York, Berlin, etc., 2001. Springer-Verlag.
17. H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4):469–476, 1975.
18. G. J. Myers. *The Art of Software Testing*. John Wiley & Sons, New York, 1979.
19. S. Oaks. *Java Security*. O'Reilly & Associates, Sebastopol, CA, USA, 1998.
20. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, Berlin, etc., 1985.
21. M. Tork Roth and P. Schwarz. Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 266–275, Athens, Greece, August 1997.
22. International Organization for Standardization. Database Language SQL. Document ISO/IEC 9075:1999, 1999.
23. M. Stonebraker and L. Rowe. The Design of POSTGRES. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 340–355, Washington, USA, 1986.
24. G. Weikum. The Web in 2010: Challenges and Opportunities for Database Research. In *Informatics - 10 Years Back. 10 Years Ahead.*, volume 2000 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2001.
25. K. Wilkinson, P. Lyngbaek, and W. Hasan. The Iris Architecture and Implementation. *IEEE Trans. Knowledge and Data Engineering*, 2(1):63–75, March 1990.