AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Managing data availability and integrity in federated cloud storage

Zarządzanie dostępnością i integralnością danych w sfederowanych zasobach chmury obliczeniowej.

Krzysztof Styrc

Promotor: dr inż. Marian Bubak (AGH)
Konsultant: Piotr Nowakowski (ACC Cyfronet)

Krótko o projekcie VPH-Share

Projekt VPH-Share ma na celu:

- stworzenie platformy typu cloud dla potrzeb sektora medycznego
- umożliwiającej udostępnianie i wymianę danych oraz wiedzy,
- której jednym z wymagań jest ich wiarygodność i integralność.

Komponent Data Reliability and Integrity(DRI) ma za zadanie:

- okresowe sprawdzanie wiarygodności i integralności danych,
- umożliwiać replikację danych na różne platformy cloudowe,
- śledzić historię i pochodzenie plików z danymi.

Wiarygodność i integralność pliku

Temat dobrze zbadany:

- funkcje skrótu(MD5, SHA-1),
- Message Authentication Code(MAC),
- Error Correcting Code(ECC).

Nowe wyzwania - cloud storage:

- przechowywanie bardzo dużych zbiorów danych,
- przechowywanie danych na obcych zasobach,
- brak gwarancji wiarygodności i integralności.

Wiarygodność i integralność pliku

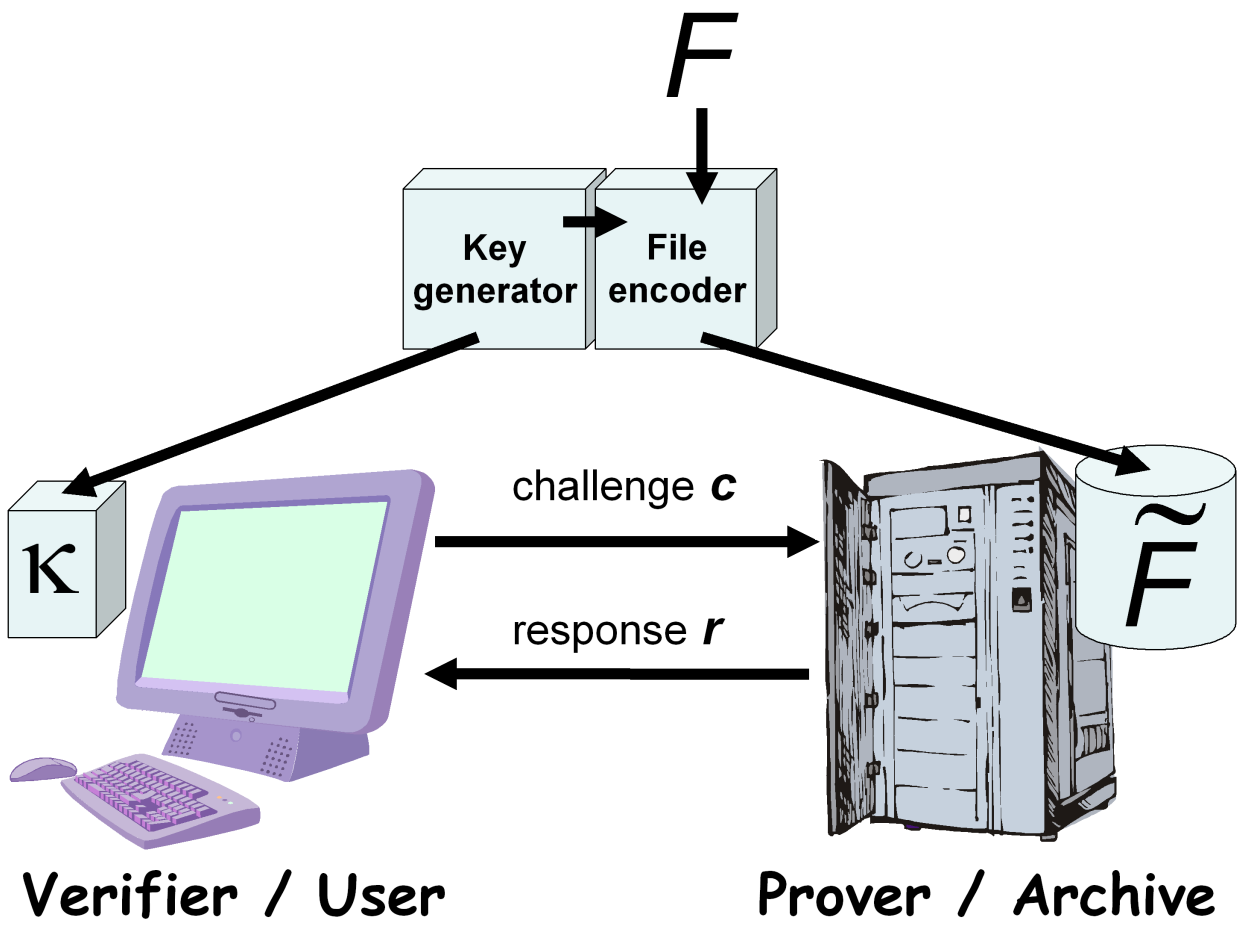
Konsekwencje przechowywania danych w chmurze:

- brak gwarancji wiarygodności i integralności,
- nieefektywność zewnętrznej walidacji - duże zbiory danych,

Proof of Retrievability(POR):

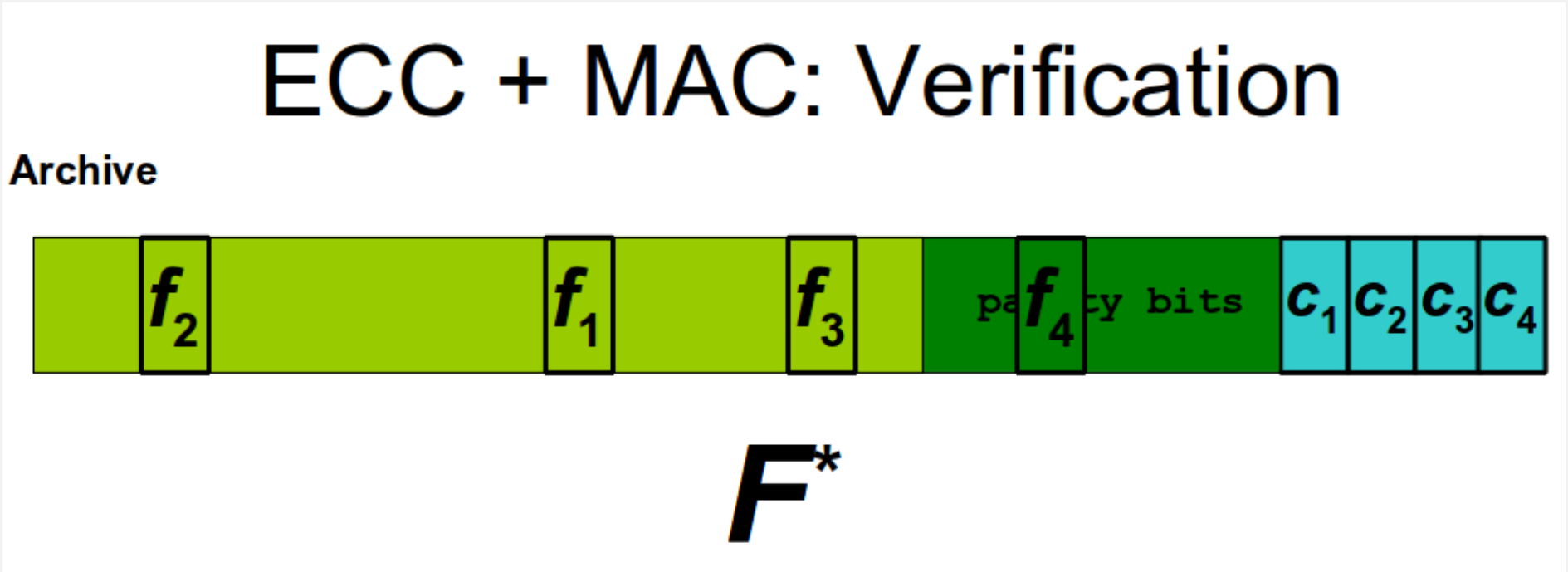
- walidacja tylko po części zawartości pliku,
- plik jest przechowywany w formie zmodyfikowanej,
- poprawność oparta o zastosowanie funkcji skrótu, MAC i ECC,
- sumy kontrolne w większości wbudowane w plik(modyfikacja).

Ogólny schemat systemu POR



Modyfikacja ECC + MAC

Zmodyfikowany plik F^* może wyglądać następująco:



Wiarygodność i integralność w VPH-Share

Wymagania platformy VPH-Share:

- przechowywanie plików w postaci niezmodyfikowanej,
- sumy kontrolne przechowywane w rejestrze Atmosphere, a nie zawarte w pliku - rejestr ma ograniczone rozmiary!

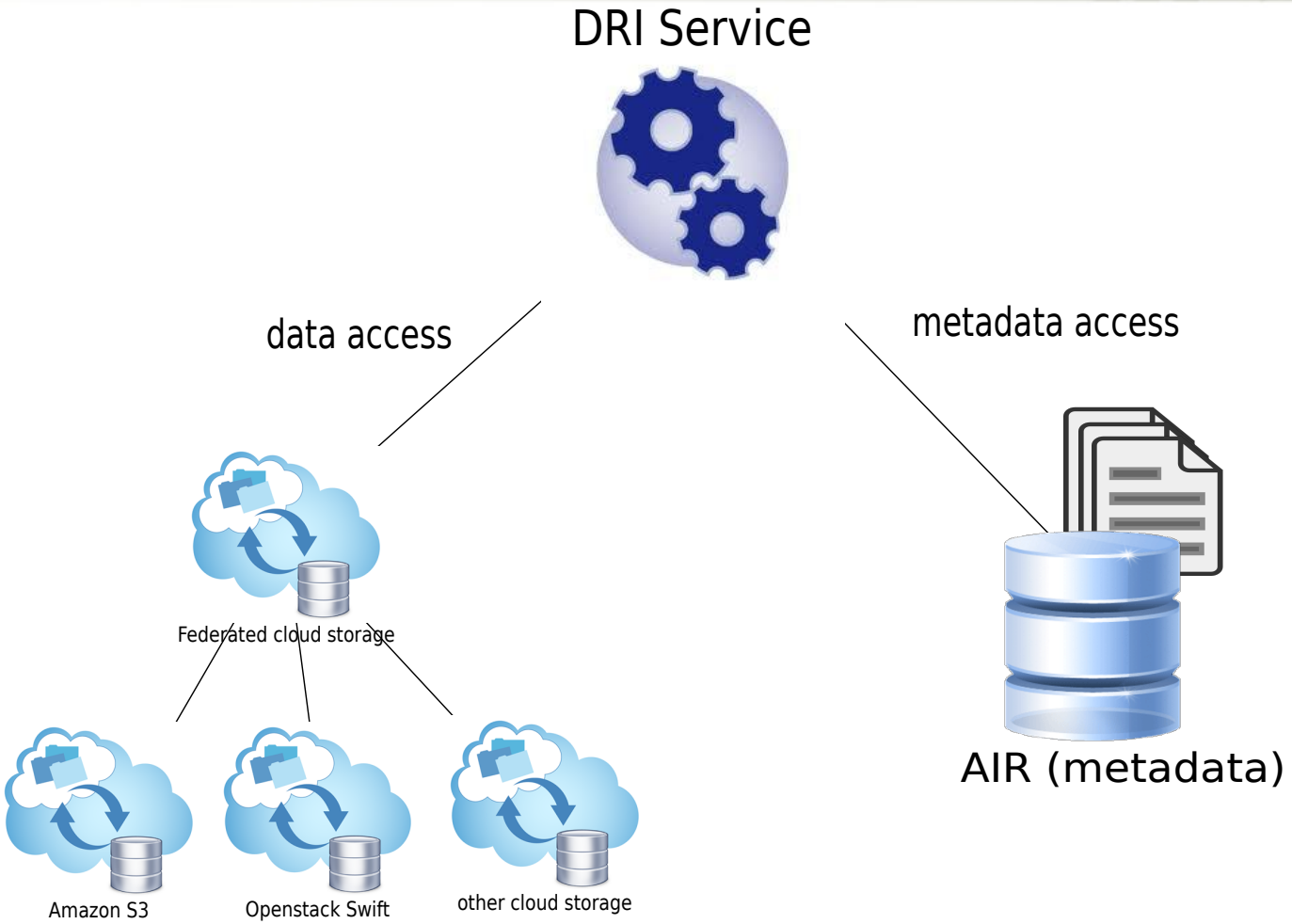
Cel pracy:

- opracowanie algorytmu walidacji w oparciu o podejście POR, który uwzględniałby powyższe wymagania i ograniczenia,
- porównanie różnych algorytmów i wybranie tego, który da najlepszą wykrywalność błędów,
- implementacja komponentu walidacji DRI w oparciu o ten algorytm.

Bibliografia

- VPH-Share Deliverable 2.1
- VPH-Share Deliverable 2.2
- PORs: Proofs of Retrievability for Lare Files - Juels, Kaliski,
- HAIL: A High Availability and Integrity Layer for Cloud Storage - Juels, Bowers, Oprea
- Venus: Verification for Untrusted Cloud Storage - wielu autrów

Aktualna architektura



Opis serwisów i integracji

Opis serwisów:

- DRI Service – serwis do walidacji danych,
- Swift BlobStore – przechowuje dane z zapewnieniem replikacji,
- Metadata Registry – baza metadanych obiektów z BlobStore,

Integracja:

- Swift BlobStore – udostępnia interfejs REST API, istnieje świetna biblioteka JClouds dostępu do Cloud Storage,
- Metadata Registry – wewnętrzna implementacja projektu VPH-Share, udostępnia interfejs REST API, integracja możliwa za pomocą JAX-RS.

Algorytm walidacji

Algorytm walidacji:

1. pobierz metadane zbioru danych(nazwa kontenera, nazwy plików, wartości skrótów, itp) z Metadata Registry,
2. dla każdego pliku ze zbioru pobierz potrzebne dane do obliczenia wartości skrótu ze Swift BlobStore,
3. dla każdego pliku porównaj otrzymaną wartość skrótu z wartością zapisaną w metadanych.

Ocena wydajności systemu

Przykładowy zbiór:

- zbiór zawiera $64K$ plików,
- średnia wielkość pliku $\sim 0.5MB$,
- wielkość zbioru $\sim 32GB$,

Obliczenia:

- obliczanie skrótow MD5/SHA1 ma większą przepustowość niż transfer sieciowy,
- największy narzut – transfer sieciowy,
- załóżmy transfer $5MB/s$,
- czas przesyłu $\sim 1.8h$!

Problemy, problemy

Problemy, problemy:

- pożądane byłoby pobranie wszystkich potrzebnych do walidacji fragmentów w jednym żądaniu HTTP,
  np. zwróć mi bajty 5-10, 45-50, 95-100, itd.
- rozwiązanie: użycie nagłówka HTTP Range w celu określenia żądanych fragmentów pliku,
  np. Range:5-10,45-50,95-100,...
- niestety: Openstack Swift aktualnie nie wspiera w pełni nagłówka Range, możliwe jest określenie wyłącznie pojedyńczego fragmentu,
  np. Range:5-10

Problemy, Problemy

Pomysły na rozwiązanie

- zwrócić się do twórców Openstack Swift o implementację pełnego wsparcia nagłówka HTTP Range – niestety przy obsłudze HTTP Openstack korzysta z biblioteki *Webob*, a znowu jej twórcy stwierdzili, że nie znają sensownego zastosowania pełnego nagłówka HTTP Range poza opublikowanym atakiem DDoS na Apache Server,
- wysyłanie osobnych żądań HTTP o każdy fragment – nieskalowalne, zalewa serwer żądaniami, w znacznym stopniu ograniczyłoby to liczbę fragmentów na podstawie których obliczany jest skrót pliku,
- ściąganie całości pliku – jak wykazała moja krótka analiza wydajnościowa, nieakceptowalne.

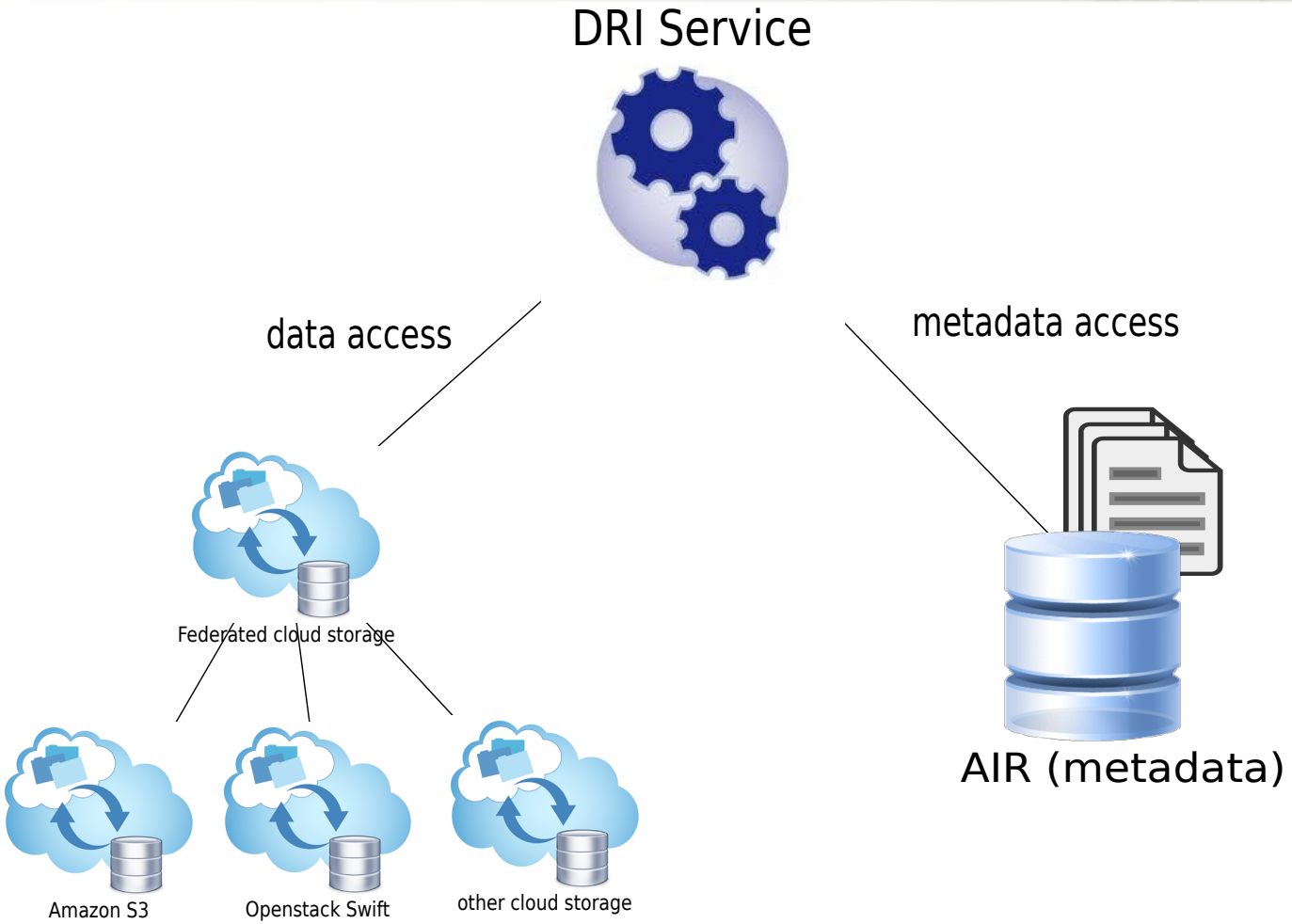
Managing data availability and integrity in federated cloud storage

Zarządzanie dostępnością i integralnością danych w sfederowanych zasobach chmury obliczeniowej.

Krzysztof Styrc

Promotor: dr inż. Marian Bubak (AGH)
Konsultant: Piotr Nowakowski (ACC Cyfronet)

Architektura DRI



Wymagania projektu

Wymagania funkcjonalne:

- periodyczne sprawdzanie dostępności i integralności danych (walidacja),
- przeprowadzenie walidacji na żądanie użytkownika,
- replikacja danych w federacji cloudów,
- śledzenie historii i pochodzenia danych binarnych.

Wymagania projektu

Wymagania niefunkcjonalne:

- efektywność walidacji danych,
- bezpieczeństwo serwisu DRI,
- skalowalność rozwiązania,
- możliwość konfiguracji parametrów.

Przypadki użycia

Użytkownik dodaje zbiór do walidacji

1. pobranie metadanych zbioru danych z rejestru AIR,
2. obliczenie sum kontrolnych,
3. aktualizacja zbioru danych w rejestrze AIR,
4. dodanie zbioru do periodycznej walidacji,

w przypadku błędu zwrócenie go użytkownikowi.

Przypadki użycia

Użytkownik dokonuje walidacji zbioru danych:

1. pobranie metadanych zbioru danych z rejestru AIR,
2. sprawdzenie dostępności i integralności zbioru danych,
3. wysłanie informacji o statusie walidacji do serwisu notyfikacji.

Przypadki użycia

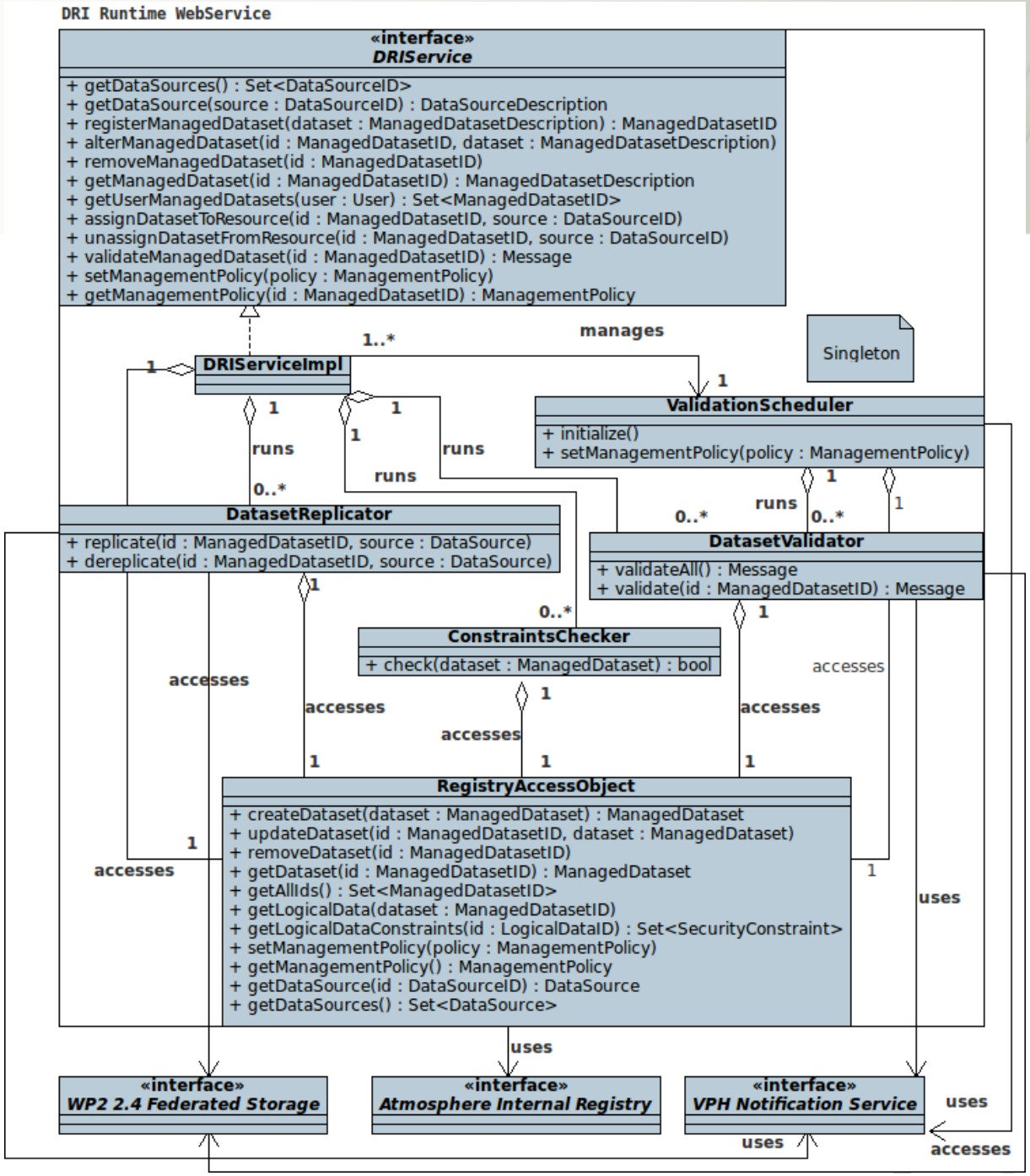
Użytkownik dokonuje replikacji zbioru danych:

1. pobranie metadanych zbioru danych z rejestru AIR,
2. sprawdzenie ograniczeń zbioru danych,
3. replikacja zbioru danych na wybrany cloud,
4. wysłanie informacji o zakończeniu do serwisu notyfikacji,

w przypadku błędu wysłanie błędu do serwisu notyfikacji.

Interfejs serwisu DRI

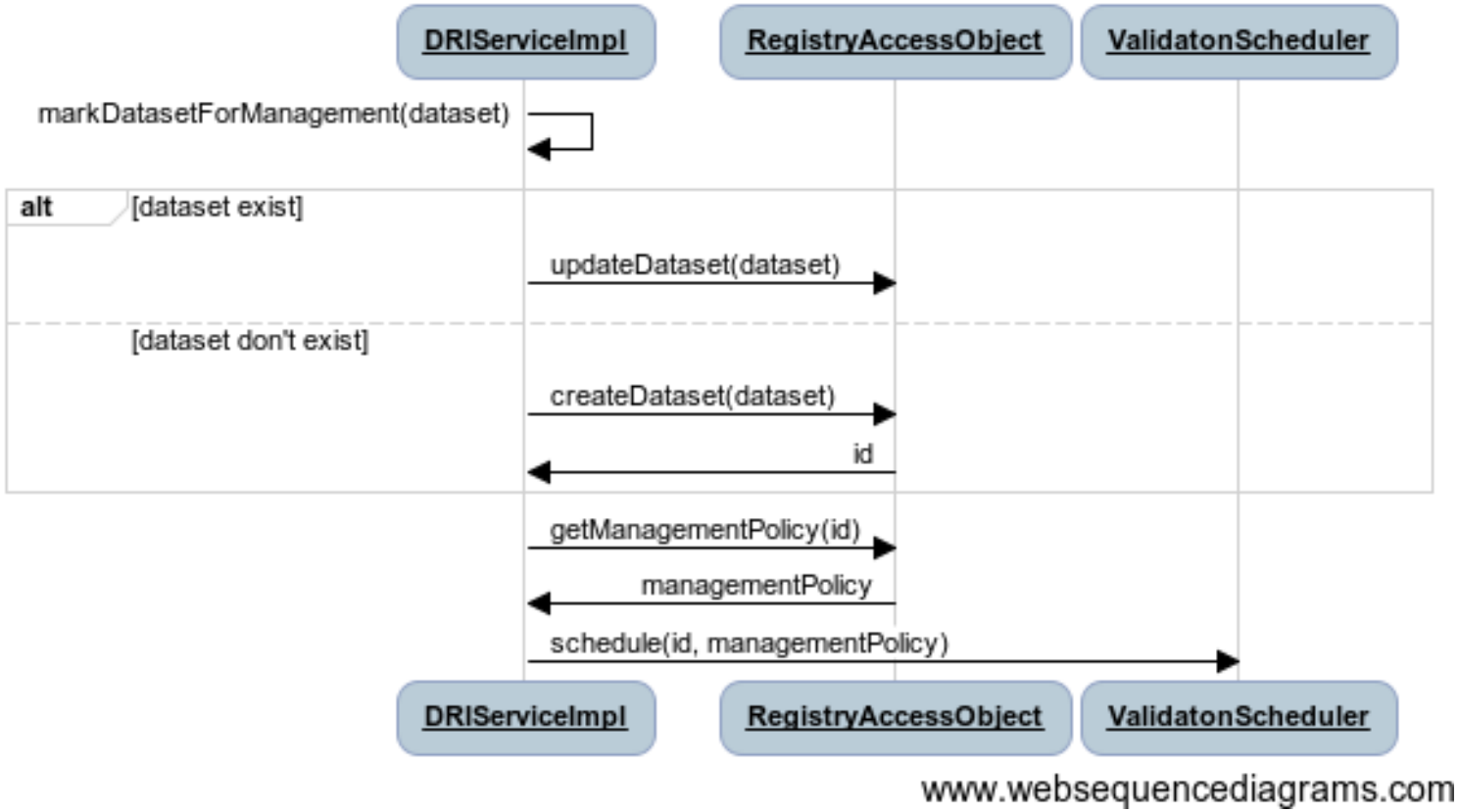
«interface» ***DRIService***
+ getDataSources() : Set<DataSourceID> + getDataSource(source : DataSourceID) : DataSourceDescription + registerManagedDataset(dataset : ManagedDatasetDescription) : ManagedDatasetID + alterManagedDataset(id : ManagedDatasetID, dataset : ManagedDatasetDescription) + removeManagedDataset(id : ManagedDatasetID) + getManagedDataset(id : ManagedDatasetID) : ManagedDatasetDescription + getUserManagedDatasets(user : User) : Set<ManagedDatasetID> + assignDatasetToResource(id : ManagedDatasetID, source : DataSourceID) + unassignDatasetFromResource(id : ManagedDatasetID, source : DataSourceID) + validateManagedDataset(id : ManagedDatasetID) : Message + setManagementPolicy(policy : ManagementPolicy) + getManagementPolicy(id : ManagedDatasetID) : ManagementPolicy


DRI Runtime WebService
«interface»
DRIService
+ getDataSources() : Set<DataSourceID>
+ getDataSource(source : DataSourceID) : DataSourceDescription
+ registerManagedDataset(dataset : ManagedDatasetDescription) : ManagedDatasetID
+ alterManagedDataset(id : ManagedDatasetID, dataset : ManagedDatasetDescription)
+ removeManagedDataset(id : ManagedDatasetID)
+ getManagedDataset(id : ManagedDatasetID) : ManagedDatasetDescription
+ getUserManagedDatasets(user : User) : Set<ManagedDatasetID>
+ assignDatasetToResource(id : ManagedDatasetID, source : DataSourceID)
+ unassignDatasetFromResource(id : ManagedDatasetID, source : DataSourceID)
+ validateManagedDataset(id : ManagedDatasetID) : Message
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy(id : ManagedDatasetID) : ManagementPolicy
manages
1..*
1
Singleton
DRIServiceImpl
ValidationScheduler
+ initialize()
+ setManagementPolicy(policy : ManagementPolicy)
runs
DatasetReplicator
+ replicate(id : ManagedDatasetID, source : DataSource)
+ dereplicate(id : ManagedDatasetID, source : DataSource)
DatasetValidator
+ validateAll() : Message
+ validate(id : ManagedDatasetID) : Message
ConstraintsChecker
+ check(dataset : ManagedDataset) : bool
accesses
RegistryAccessObject
+ createDataset(dataset : ManagedDataset) : ManagedDataset
+ updateDataset(id : ManagedDatasetID, dataset : ManagedDataset)
+ removeDataset(id : ManagedDatasetID)
+ getDataset(id : ManagedDatasetID) : ManagedDataset
+ getAllIds() : Set<ManagedDatasetID>
+ getLogicalData(dataset : ManagedDatasetID)
+ getLogicalDataConstraints(id : LogicalDataID) : Set<SecurityConstraint>
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy() : ManagementPolicy
+ getDataSource(id : DataSourceID) : DataSource
+ getDataSources() : Set<DataSource>
uses
«interface»
WP2 2.4 Federated Storage
«interface»
Atmosphere Internal Registry
«interface»
VPH Notification Service

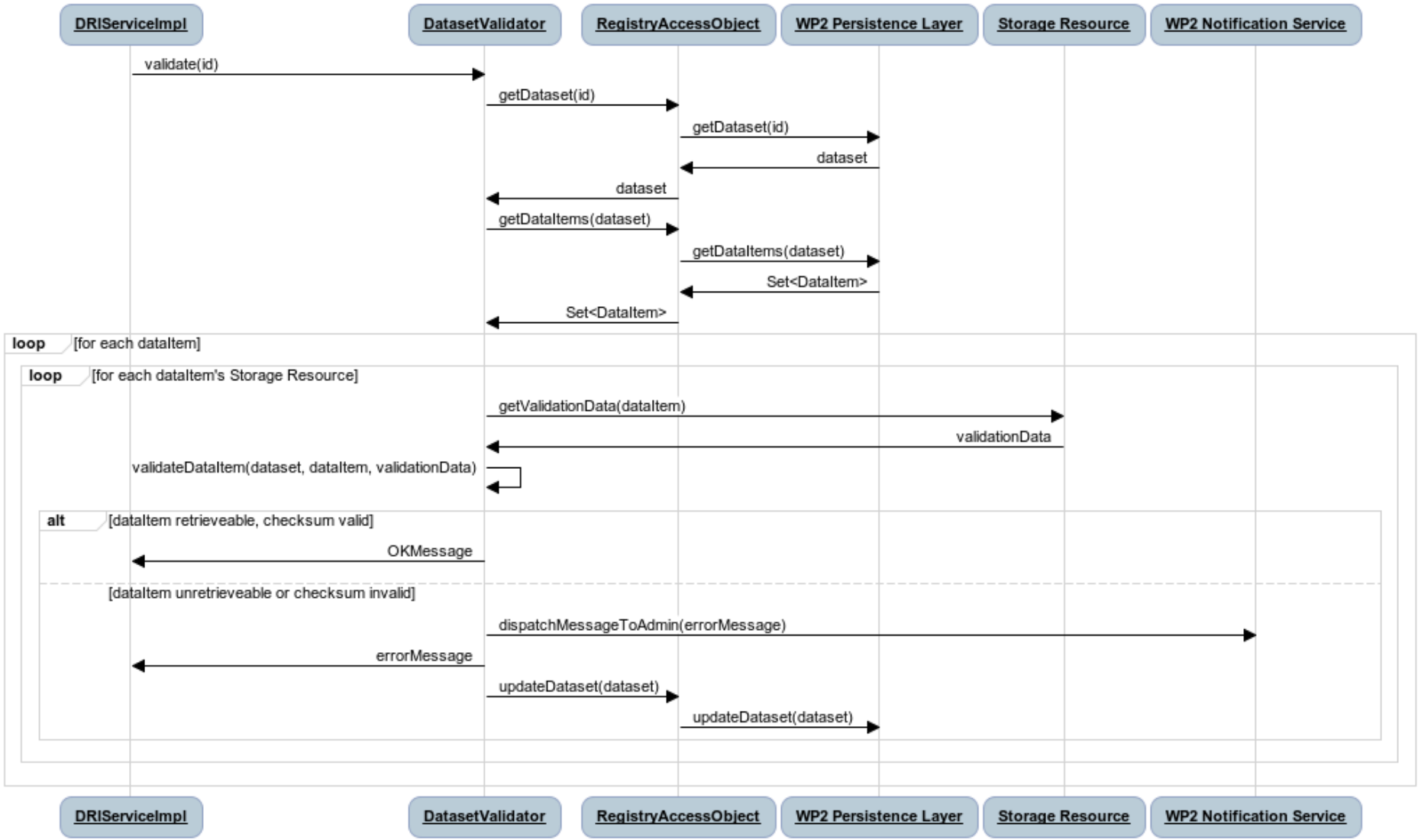
Używane technologie:

- Java - język implementacji,
- JAX-RS i Jersey - interfejs REST serwisu,
- Jclouds - jednolity dostęp do różnych platform cloudowych,
- Google Guava i Guice, Apache Commons,
- Apache Tomcat, Maven, SVN.

Użytkownik dodaje zbiór do walidacji

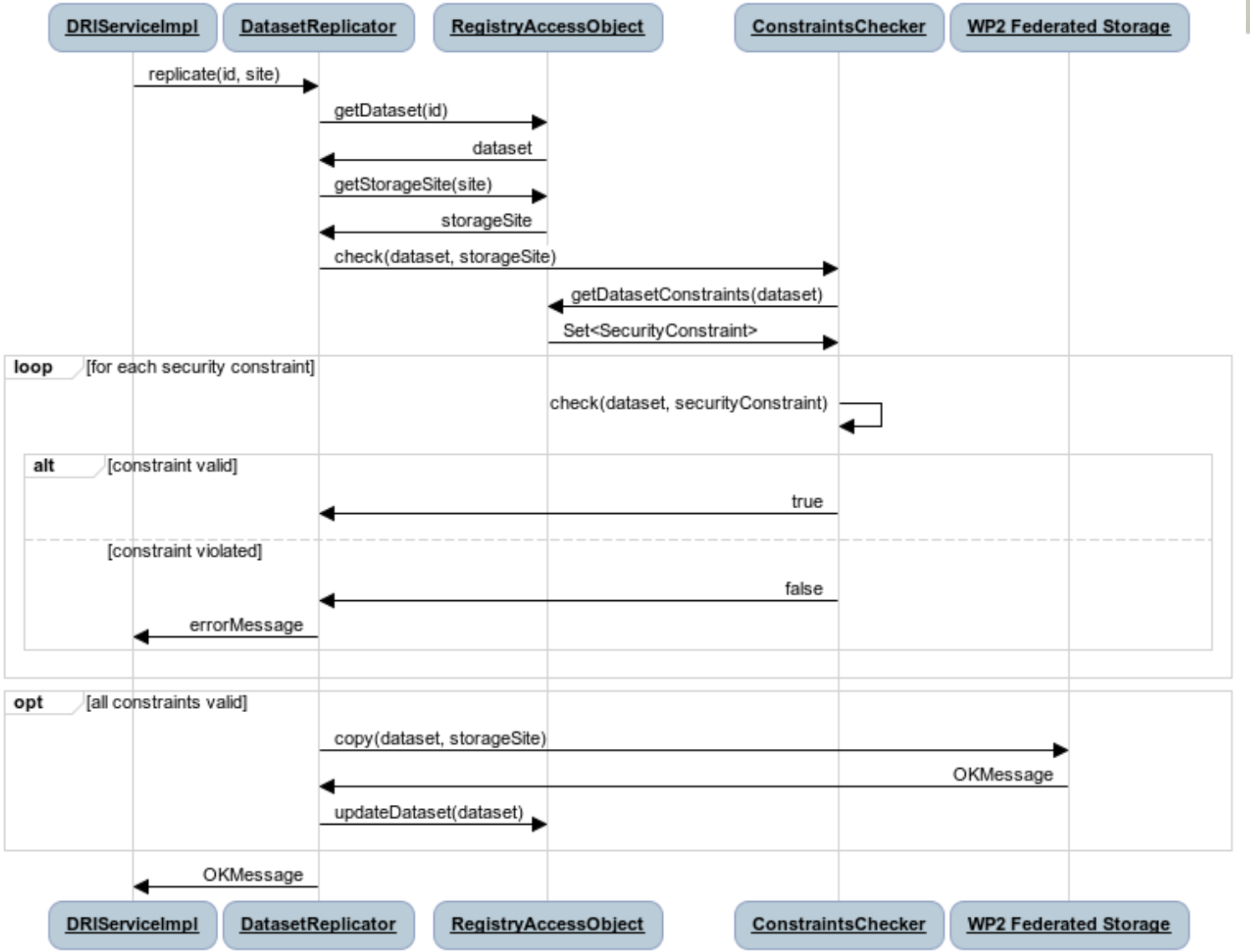


Użytkownik dokonuje walidacji zbioru danych



www.websequencediagrams.com

Użytkownik dokonuje replikacji zbioru danych



www.websequencediagrams.com

Stan prac:

- periodyczna walidacji zbiorów danych,
- walidacja danych na żądanie,
- walidacja oparta o SHA-1,
- notyfikowanie o statusie walidacji,
- gotowa integracja z resztą projektu VPH-Share,
- **czas na implementację efektywnych algorytmów walidacji.**