
Technical White Paper

VYRE Unify

VYRE

July 31, 2006

Table of contents

- 1. Introduction.....2**
- 2. Architecture.....2**
 - 2.1 System Components.....2
 - 2.1.1 Requirements4
- 3. Deployment Strategies.....4**
 - 3.1 Dynamic Delivery6
 - 3.2 Selective Content Replication (SCORE)6
 - 3.3 Caching7
 - 3.3.1 Static delivery8
- 4. Extending and Integrating8**
 - 4.1 Plug-ins and Extensions8
 - 4.1.1 Portlets8
 - 4.1.2 Realms9
 - 4.1.3 e-Commerce Payment Mechanisms.....9
 - 4.1.4 Skins and localization.....9
 - 4.2 Content Integration9
 - 4.2.1 Feeds9
 - 4.2.2 Web Services10

1. Introduction

Unify from VYRE is a complete, flexible framework for the creation and management of dynamic content driven websites and web based applications. Rapidly implemented and easily adopted by users of all skill levels, Unify is a feature rich architecture which offers versatility and performance whilst providing a cost effective way of managing both structured and unstructured content.

This white paper gives an overview of the architecture of Unify, and the technology behind it. We'll show you how different strategies for deploying the system, and how Unify can be extended with plug-ins and integrated into existing environments.

2. Architecture

VYRE Unify is a set of lightweight **Java Enterprise Edition (Java EE)** web applications, and a Digital Asset Processing Server (DAPS). It is accessible through a 100% browser based interface (IE, Firefox, Safari, etc.). It runs on top of a relational database, but binary files and structured content files (XML) are stored in the file system, with all text extracted and indexed for fast and easy retrieval. Any Java EE compliant application server can be used to run Unify, and in fact only a servlet container (such as Apache Tomcat) is needed. Also, an object persistence framework is used to abstract the details of the underlying database, which means that VYRE Unify can run on any major relational database. The system is standards based and highly object-oriented and modular, so it can easily be extended if needed. Let's recap:

- Platform independent
- Database independent
- Binary files and structured content files stored in the file system
- Runs on any Java application server or servlet container
- 100% Accessible via a browser
- Standards-based and extendable

2.1 System Components

VYRE consists roughly of the following parts:

- Backend: This is the administration application where content types are defined and sites are created.
- Delivery: Lightweight applications for delivering sites to end users.

- **DAPS:** Digital asset processing server, handles file transformations.
- **Content repository:** A central server that stores the content.

This is illustrated in the following diagram. Note that this diagram only serves to show the components involved; deployment strategies will be discussed in next section.

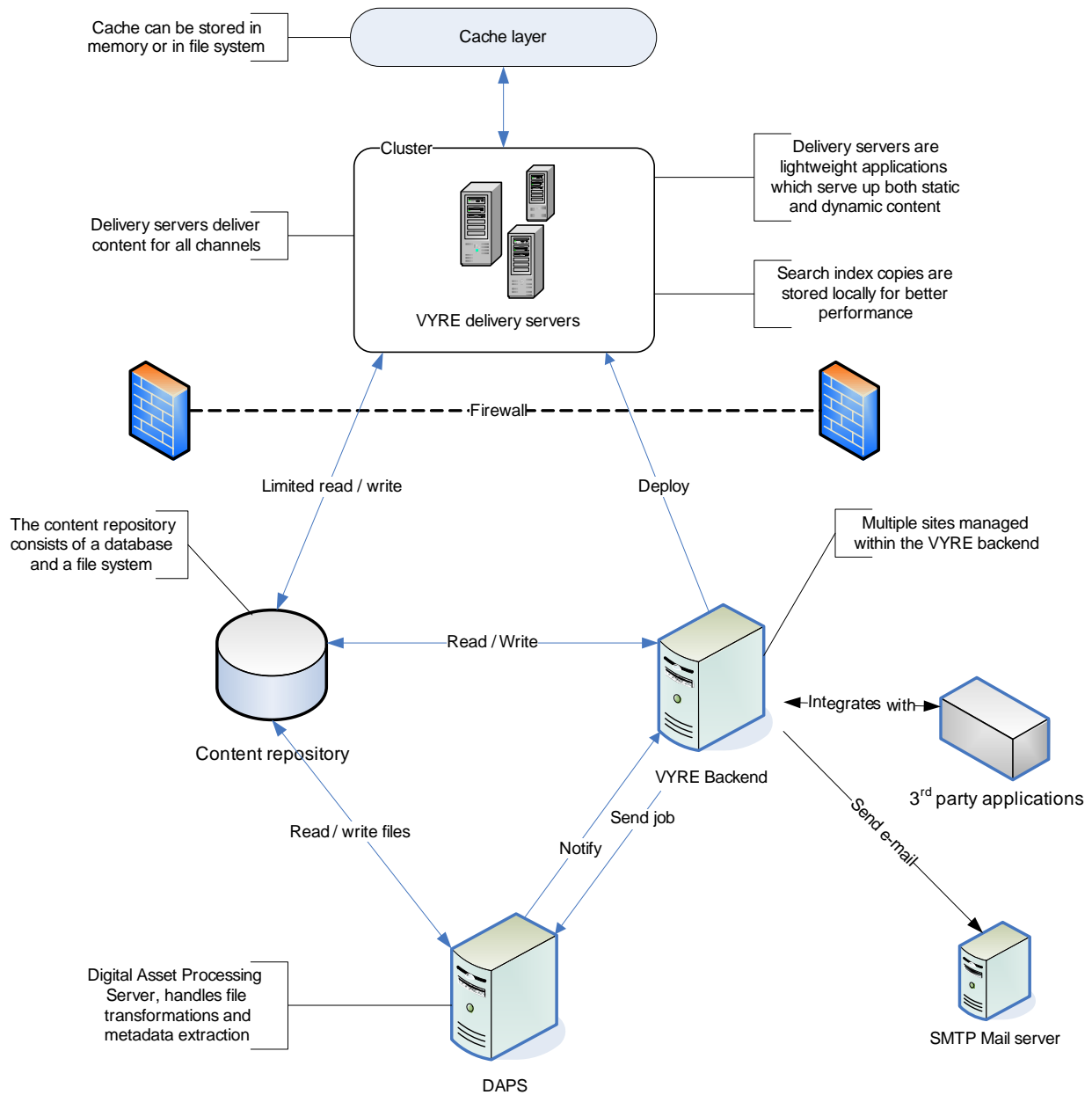


Figure 1: VYRE Unify system components

2.1.1 Requirements

The following requirements apply:

- **Unify Backend server:** The backend is a Java web application which runs on any Java servlet container or application server.
- **Delivery server:** The delivery application is a Java web application which runs on any Java servlet container or application server.
- **DAPS:** The DAPS is a .NET service, which needs to run on a Windows server. Communication with DAPS is done via jobs in the database.
- **Database:** This can be any major database. Unify is developed and tested with Oracle and MySQL, but has also been tested with other databases such as Microsoft SQL server and PostgreSQL.
- **File server:** This is a central file system, accessible by all delivery servers (and backend) in the cluster. Usually implemented as NFS mounted from a cluster onto a SAN, or similar.
- **Mail server:** Unify needs access to a mail server to send out email notifications to users.

For detailed software and hardware requirements, refer to the “VYRE Unify Setup Requirements” document.

3. Deployment Strategies

There are two main deployment strategies that can be employed in VYRE Unify:

- **Dynamic delivery:** One cluster, the main cluster, is used.
- **Selective Content Replication (SCORE):** in addition to the main cluster, a set of SCORE clusters are employed.

Before we continue, it is important to define the concepts involved.

- A **main cluster** is a set of one admin server and one or more delivery servers. An **admin server** is a server running the VYRE Unify administration application, where data structures are defined and sites are managed. A **delivery server** is a server running a lightweight delivery application which serves up sites.
- A **SCORE cluster** is a set of one or more read-only delivery servers that serve up a subset of sites and content from the repository. A **read-only delivery server** only serves up read-only sites. A *read-only site* is a site that only contains read-only portlets. *Read-only portlets* can not write to the native Unify content/user repository, and can not personalize their output to individual users.

Most deployments will use dynamic delivery. SCORE clusters are useful when a completely decoupled delivery mechanism is needed. Each SCORE cluster has a local repository, which is a subset of the main repository. The following diagram gives an overview clusters in VYRE Unify. Note that each deployment MUST have a main cluster, but SCORE clusters are optional.

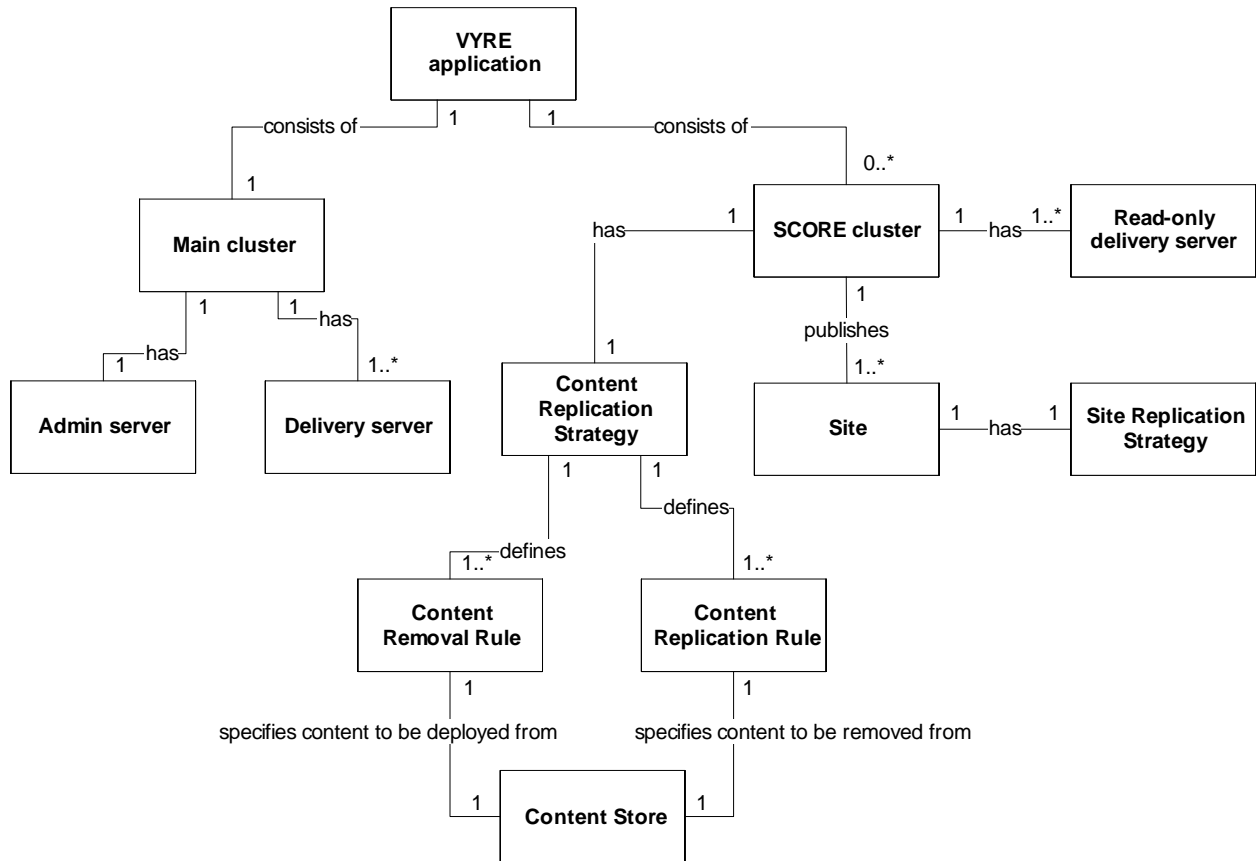


Figure 2: VYRE cluster domain model

3.1 Dynamic Delivery

This is the default deployment strategy. It involves deploying only one cluster, the main cluster. This is the most flexible strategy, since it allows delivery servers to write to the central content repository. This strategy is therefore useful for sites such as internal content applications, intranets, and low to medium load sites.

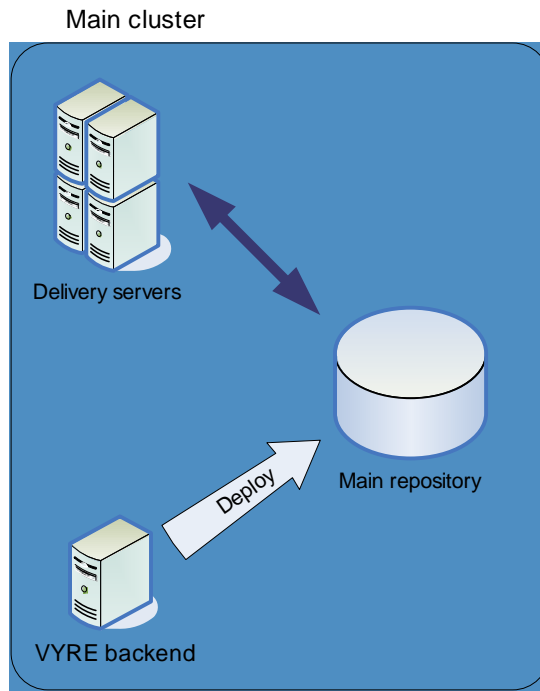


Figure 3: Dynamic delivery

3.2 Selective Content Replication (SCORE)

Selective content replication (SCORE) is the process of selecting a subset of the content repository and securely transferring it from the main cluster to a SCORE cluster. Rules (based on metadata attributes) are defined which govern which content is replicated to (or removed from) a specific SCORE cluster.

This is a “de-coupled” delivery architecture for the deployment of web applications. These applications are designed and built on the main cluster, and then generated and replicated to SCORE clusters. Once replicated, a web application is independent of the main cluster (which can therefore be disconnected) and does not need to perform repository API calls through the VYRE Unify backend.

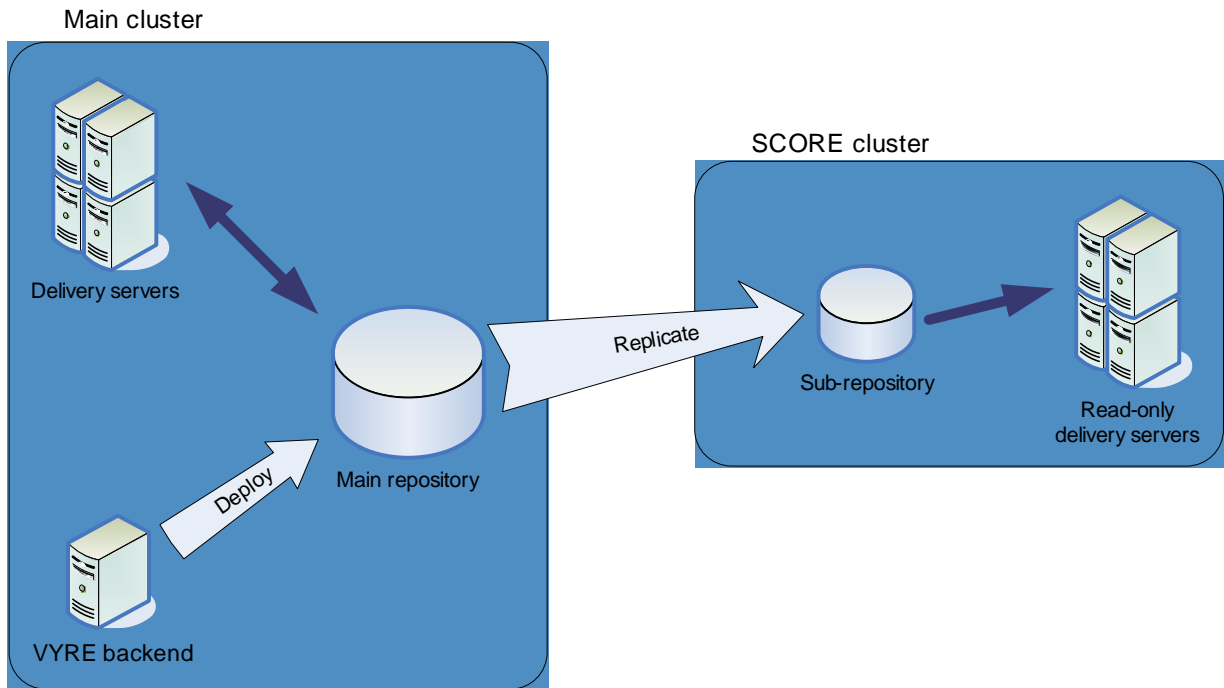


Figure 4: The SCORE strategy

The SCORE strategy is very scalable, and gives high performance. It is however important to stress that an application deployed on a SCORE cluster does not have write access to the repository. SCORE clusters can be distributed to the edge of the network, so SCORE delivery mechanism is therefore good for high-load high-availability public websites, but not for intranets.

3.3 Caching

VYRE Unify has a powerful multi-level cache support. Caching can be fully utilized independent of whether a dynamic delivery or SCORE delivery is being used. It offers three levels of caching:

- Page-level caching: caching a whole page.
- Content area caching: caching an individual content area on a page.
- Portlet caching: caching an individual portlet in a specific content area on a page.

All three levels can be employed at the same time, with different expiry times and triggers. So for example, you could build a page where the navigation areas are stored in permanent cache, the main content area is not cached at all, and the whole page uses a dynamic cache which is flushed every Sunday at midnight.

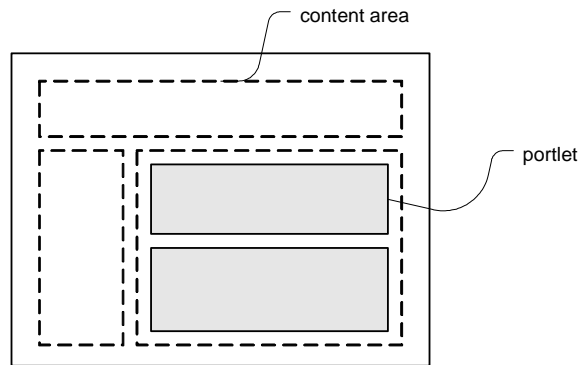


Figure 5: A page with content areas and portlets

Apart from content caching, we employ a robust cluster-safe database cache layer which stores business objects according to LRU access.

3.3.1 Static delivery

VYRE also offers the ability to generate and deploy “static” pages, rather than dynamic JSPs. In this case, the page itself is “dead”, no processing is performed before serving it up. However, VYRE still employs lightweight filters which enforce permissions and access rules, which make sure that restricted static pages can never be called directly.

4. Extending and Integrating

Unify makes use of a set of interfaces and APIs to allow integration with 3rd party systems. The technology at our disposal are a mixture of Open Source tools that are mature or rapidly maturing and recognized as standard methodologies, widely adopted by leading software vendors as part of their long term strategy to enable integration with their applications.

A summary is provided below of interfaces and API’s for extending VYRE, and accessing the content repository. This is useful for integrating 3rd party applications into Unify and vice versa.

4.1 Plug-ins and Extensions

4.1.1 Portlets

Portlets are pluggable user interface components that produce fragments of markup code that are aggregated into a portal page. Unify has a built-in portlet container utilizing the Java Portlet Specification (JSR-168),

allowing users to develop portlets that access 3rd party software to deploy content to web pages published with Unify. Users can deploy their Java Portlet Applications into Unify, reusing existing portlets, and/or developing new portlets to publish content from existing legacy systems. Unify will automatically detect new portlet applications, and make them available to end users. Users can then point-and-click to add portlets to pages.

For detailed information on how to deploy portlet applications in Unify, refer to the “Deploying portlet applications in VYRE Unify” document.

4.1.2 Realms

For user and user group management, Unify supplies a unique extendable User Realm API, which allows users to implement bespoke realms connecting to any external user resources. Unify comes with three realm implementations out of the box, JDBC, LDAP and Consolidated Realm.

With the extendable User Realm API you can allow users in your existing user database to log into sites published with Unify.

4.1.3 e-Commerce Payment Mechanisms

Unify’s e-Commerce module ships with an API for integration with Payment Service Providers (PSP) to transact payments online. PSP’s can be either remote or integrated. With Integrated PSP, the customer details are captured within your website and the transaction processed via a third-party library running on the Unify server. Remote PSP’s however, send the customer off to third-party website for payment processing and clearing.

4.1.4 Skins and localization

Unify comes with instructions on how to change the look-and-feel of the administration interface (this is accomplished with a simple CSS). Also, new translations of the system’s interface can be created using TMX (Translation memory exchange) files.

4.2 Content Integration

4.2.1 Feeds

VYRE Unify has syndication and aggregation abilities that are useful for content integration between sites. Unify has the capability to publish localized, optionally filtered, content feeds on sites. Supported output

formats are RSS, RDF, and Atom. End users are allowed to create a custom feed to subscribe to personally requested content.

Unify also ships with portlets that can aggregate feeds from remote sites. Feeds can be filtered using static constraints, or user based constraints. Supported input formats are RSS, RDF, and Atom. Custom XML formats can also be parsed, but this requires a custom XSL style sheet to be written.

4.2.2 Web Services

4.2.2.1 Content web services

Through content web services, 3rd party systems can publish content managed by Unify. Unify's content web services provide a web service interface for searching and retrieving content items held in the data and file stores in the Content module.

4.2.2.2 User Web Services

User web services allow 3rd party systems to retrieve user information and update it through a web service interface. This allows other systems to automatically synchronize its user database with a user database in Unify and authenticate them, or alternatively to synchronize a User Realm in Unify with its own user database.