D3.1
GloNet Platform Design Specification

Edited by
Markus Bauer, CAS
June 2012

Project funded by the European Commission under the
ICT – Factories of the Future Programme

Contract Start Date: 1 Sep 2011  Duration: 36 months
This report documents the architecture of the GloNet platform. The GloNet platform is aimed at providing a framework for the construction of scalable, extensible cloud-based solutions for the collaborative development and operation of highly customized and service-enhanced products.

The GloNet platform uses a framework-based approach: it defines an application architecture that serves as a blueprint to implement GloNet solutions and it provides a number of components that implement the basic building blocks if this architecture. The GloNet platform is extensible; therefore it also provides customization mechanisms as well as extension and integration mechanisms for additional modules and services that may complement the basic features of the platform. These mechanism work on all layers of the platform architecture, i.e. the data layer, the logic layer and the user interface layer are customizable and extensible.

The extensible nature of the GloNet platform imposes a number of specific requirements, which are complemented by a number of requirements originating from the design as a cloud-based platform.

The report summarizes these requirements and presents a suitable architecture to address these requirements. It makes this architecture concrete by defining the basic building blocks of the architecture and describing their responsibilities and interfaces.

These building blocks provide important basic services (including user interfaces) for collaborative applications, especially with respect for building up a repository of information and knowledge assets.

Additionally, the platform provides mechanisms for the integration of external systems and services: a plugin based mechanism for integrating externally deployed web service, a mashup-like mechanism for integrating web-based application fragments and a framework for data synchronization for the integration of mass data that is already available in legacy systems operated by some of the partners in a GloNet collaboration scenario.
# TABLE OF CONTENTS

## PROJECT-RELATED SUMMARY

1. INTRODUCTION .......................................................................................................................... 6

2. Basics of Cloud Platforms ........................................................................................................... 8
   2.1 Definition of Cloud Computing .............................................................................................. 8
   2.2 Service models ........................................................................................................................ 8
   2.3 Characteristics of cloud solutions ........................................................................................... 9

3. Design Goals ................................................................................................................................ 14
   3.1 Modularity and extensibility ................................................................................................. 14
   3.2 Multi-tenancy ........................................................................................................................ 15
   3.3 Scalability and Availability ................................................................................................... 15
   3.4 Security and Trust ................................................................................................................. 16
   3.5 Network access ...................................................................................................................... 16

4. Architecture ................................................................................................................................ 17
   4.1 Layered architecture .............................................................................................................. 17
   4.2 System walkthrough .............................................................................................................. 17
   4.3 Support for a metadata based configuration .......................................................................... 18
   4.4 Support for dynamic modules ............................................................................................... 19
   4.5 Deployment architecture ....................................................................................................... 20
   4.6 Technical infrastructure ......................................................................................................... 21
   4.7 Data layer .............................................................................................................................. 21
      4.7.1 Support for multi-tenancy .............................................................................................. 21
      4.7.2 Mapping multi-tenancy to VO concepts ........................................................................ 22
      4.7.3 Support for generic, customizable data objects ............................................................. 24
      4.7.4 Relations ........................................................................................................................ 28
      4.7.5 Implementation structure ............................................................................................... 30
      4.7.6 Query language .............................................................................................................. 31
   4.8 Logic layer ............................................................................................................................. 31
      4.8.1 Interface layer ................................................................................................................ 32
      4.8.2 Business operations ....................................................................................................... 33
      4.8.3 Support for users, roles and permissions ....................................................................... 37
      4.8.4 Documents ..................................................................................................................... 41
      4.8.5 Journaling ........................................................................................................................ 43
   4.9 Presentation layer .................................................................................................................. 44
   4.10 Security mechanisms .......................................................................................................... 46

5. Integration mechanisms ............................................................................................................. 47
   5.1 Service-API ............................................................................................................................ 47
GloNet

5.1.1 Proxy-based integration ................................................................................................. 47
5.1.2 Mashup-based integration ............................................................................................. 48
5.2 OpenID .................................................................................................................................. 50
5.3 Synchronization framework .................................................................................................. 51
  5.3.1 The architecture on a bird’s eye view level ................................................................. 53
  5.3.2 Synchronization concepts .............................................................................................. 54
6 GloNet development tools .......................................................................................................... 60
7 Conclusion ................................................................................................................................... 61
8 REFERENCES ................................................................................................................................ 62
9 ANNEX ........................................................................................................................................ 64
  9.1 EIMInterface, the platform’s main façade ........................................................................ 64
  9.2 Examples for using the GloNet webservice API .............................................................. 65
PROJECT-RELATED SUMMARY

This deliverable is the first result of WP 3, which is devoted to the design and implementation of a cloud-based platform for solutions supporting the collaborative development and operation of highly customized and service-enhanced products.

The deliverable defines the architecture of the platform and describes an initial set of components and services to implement this architecture. WP 3 is conceived to be iterative. In further iterations, the platform will be refined and additional components will be integrated into the platform architecture.

WP 2 will define a number of concepts and functionalities for the collaborative development and operation of highly customized and service-enhanced products, which will be implemented by additional modules of the platform.

Additionally, WP 4 and WP 5 will also contribute additional services which will either directly be built on top of the platform or integrated with the platform by the integration mechanisms provided by the platform.

The deliverable describes the concepts of the GloNet platform, focusing on both, the extensible, framework-oriented nature of the platform, as well on its cloud-based nature. It addresses all layers of the platform, data layer, logic layer and user interface layer. The user interface layer however is covered in more detail in deliverable D 2.2.
1 INTRODUCTION

GloNet aims in supporting collaborative environments for networks of SMEs involved in developing, operating and maintaining highly customized and service-enhanced products with software solutions. The creation of such software solutions is an effort-intensive task, especially when these solutions are designed to run in the cloud and when they are required to be highly customizable to different end-user scenarios. A platform (in the sense of an application framework) may simplify the creation of software solutions.

This document specifies the technical design goals, the architecture and design of the GloNet platform. The GloNet platform is a cloud-oriented technology platform. It allows the creation and modification of scalable, extensible SaaS solutions for the collaborative development and operation of highly customized and service-enhanced products. A GloNet system is a system operated as a software-as-a-service solution (SaaS) tailored towards the end users, such as participants in a collaborative project, a virtual organization (VO) or a virtual breeding environment (VBE).

The GloNet platform uses a framework-based approach: it defines an application architecture that serves as a blueprint to implement GloNet systems and it provides a number of components that implement the basic building blocks if this architecture.

On top of the GloNet platform a number of additional components can be developed and deployed supporting specific scenarios for collaborative projects. These components will be designed to be integrated with the GloNet platform using predefined extension mechanisms provided by the platform. A GloNet system is then implemented by customizing and extending the GloNet base platform, its basic components and these advanced components.

The document starts by introducing some basic concepts about cloud computing in chapter 2. Chapter 3 then describes a number of design goals for the GloNet platform. These design goals cover aspects related to the framework-oriented nature of the platform as well as those related to the cloud-based operation mode. Chapter 4 derives the software architecture of the GloNet platform. After a general walk-through of the layered architecture, the chapter discusses the design of each major component of the architecture. Chapter 5 describes the platform’s mechanisms to extend its core functionality with external components or services. This includes the connection to legacy software systems and services using a synchronization framework. Chapter 6 briefly discusses the development tools for the GloNet platform. Chapter 7 concludes the document by summarizing the most important design decisions of the GloNet platform.
2 Basics of Cloud Platforms

This chapter introduces the most important cloud concepts in order to lay some basis for the detailed discussion of the GloNet platform architecture that happens in the rest of the document. It also briefly discusses some characteristic advantages and disadvantages of cloud solutions and relates them with GloNet.

2.1 Definition of Cloud Computing

Software that runs in the cloud has gained significant attraction in the past few years. As defined by the US National Institute of Standards and Technologies (NIST), cloud computing is “a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” (Mell and Grance 2011). In more detail, cloud solutions have five essential characteristics:

- They are provided in an on-demand self-service oriented manner, so a customer can unilaterally consume computing capabilities such as server time, network storage or application usage, without requiring human interaction with the service provider.
- They are available over the network and accessible through standard mechanisms. This usually implies that the solutions can be used anytime and anywhere using a number of different client platforms, such as desktop computers, notebooks, tablets, mobile phones.
- The provider’s computing resources are pooled to serve multiple customers using a multi-tenant model. Physical and virtual resources are dynamically assigned and reassigned according to customers demand. Examples of such resources include storage, processing time, memory, or network bandwidth.
- Capabilities offered by the solution can be elastically provisioned and released to scale rapidly with demand. To the customer, capabilities often appear unlimited and can be consumed in any quantity at any time.
- Resource usage is metered at some level of abstraction appropriate to the type of solution or service (e.g. storage, processing, bandwidth, or active user accounts). Typically resource usage is charged on a pay-per-use basis.

2.2 Service models

There are three basic service models for cloud offerings, see:

- Software as a Service (SaaS): The capability provided to the customer is to use the provider’s application running on a cloud infrastructure. The applications are accessible from various client devices such as desktop computers, notebooks, tablets, mobile phones or just using a programatic interface (API).
- Platform as a Service (PaaS): The capability provided to the customer is to deploy onto the cloud infrastructure customer-created or acquired applications created using programming languages, libraries, services and tools supported by the provider.
- Infrastructure as a Service (IaaS): The capability provided to the customer is to provision processing, storage, networks and other fundamental computing resources where the customer is able to deploy and run arbitrary software which may include operating systems and applications.
The distinction between SaaS and PaaS solutions is a blurry one. Inspecting modern SaaS solutions, they offer many capabilities of PaaS offerings: SaaS solutions often have to be customized and tailored towards customers need to a large extend. Powerful customization mechanisms often rely on configuration languages or domain specific languages. If larger portions of a SaaS system can be adapted or customized using such languages, such systems can also be seen as PaaS offerings. Salesforce’s Force.com platform is a good example – the platform started as a highly customizable customer relationship management (CRM) solution and today it’s the basis for a large number of business applications that have almost nothing in common with CRM.

### 2.3 Characteristics of cloud solutions

Cloud solutions exhibit a number of advantages and disadvantages for all stakeholders involved. Table 1 gives an overview on such characteristics of cloud solutions and relates them to GloNet.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages, problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>In general</td>
<td></td>
</tr>
<tr>
<td>• Reduced total cost of ownership (TCO) by exploiting the economies of scale principle</td>
<td>• Vendor lock-in</td>
</tr>
<tr>
<td>• Addressing new customer and user segments by “selling to the long tail”</td>
<td></td>
</tr>
<tr>
<td>• Flexible cost models (“pay-per-use”)</td>
<td></td>
</tr>
<tr>
<td>• Reduced initial costs</td>
<td></td>
</tr>
<tr>
<td>• Ease of getting started</td>
<td></td>
</tr>
<tr>
<td>• Ubiquitous, simple access through the internet</td>
<td></td>
</tr>
</tbody>
</table>
Reduced total cost of ownership: In the SaaS model, the provider hosts software applications and the associated data on central servers at the provider’s location, and the provider also supports the hardware and software with a dedicated support staff. This relieves the user (or customer) from the responsibility for supporting the software and for purchasing and maintaining server hardware for it. In essence, the provider cares for all maintenance tasks that are typically required when running software applications, including tasks like providing backups for the data,…

The provider (or software vendor), on the other hand, has to provide all these tasks instead. However, the provider can vastly benefit from the principle of economy of scale. A SaaS vendor can serve a potentially large number of customers using one single consolidated environment. For example, a SaaS application installed in a load-balanced farm of five servers may be able to support 50 medium-sized customers. This means one customer would only be responsible for a tenth of a server. A similar application installed on-premise might require each customer to dedicate an entire server to the application – perhaps even more than one, if load balancing and high availability are concerns (Chong and Carraro 2006). A similar calculation might yield for trained staff that monitors and – in case of problems – troubleshoots the systems. These effects lead to substantial cost savings over traditional models. Given fair pricing models, both customers and vendors will benefit from these cost savings.

Addressing new markets, customers and users: The aforementioned reduced operating costs per customer also allows vendors to address new markets by benefitting from selling to the “long tail” (Anderson 2004): smaller, niche customers that might have been inaccessible to traditional solution vendors due to high costs and the complexity of the IT systems involved with the traditional solutions can suddenly be targeted as well – provided that the SaaS solution can be tailored towards those niche customers.

Flexible cost models: In addition, customers may benefit from the fact that SaaS usually employs a pay-per-use model. Customers do not have to do a high initial investment in hardware and software, instead they are typically paying monthly or yearly fees depending on the number of user licenses they need and/or on the resources they consume. Especially for smaller customers, this simplifies the planning phase needed when introducing a new software solution drastically.

Reduced initial costs, ease of getting started: One of the most frequently marketed advantages of cloud computing is the short-term cost reduction that particularly SMEs can benefit from (Impetus Knowledge Center, 2010), (sgentrepreneurs, 2011). (Hazarika, 2012). The cost advantages associated with cloud computing are based on the fact that cloud computing users do not own the resources (server, storage, etc.) that produce the computing capabilities they require and pay for. Through flexible cost models (vide supra), cloud users are usually billed on a highly granular level for the usage (e.g. server hour, storage per month, etc.) and payment is made frequently over a certain period of time(Golden, Cloud CIO: The Cost Advantage Controversy of Cloud Computing, 2011). Some academics and practitioners point out that cloud computing is more expensive over time than hosting an own infrastructure, despite the flexible cost models. This assertion is particularly valid when considering the "OpEX vs. CapEx" dilemma, which assigns cloud computing costs to the operational expenditures of a business (Golden, 2012). However, this perspective on cloud expenses ignores the reality of SMEs who only have very limited resources and need maximum cash flow. The "OpEx vs. CapEx"-view on cloud computing also neglects incurring labor and maintenance costs associated with owning an infrastructure. Furthermore, the pace of technological advancement, fiercer global competition, and the concurrent increase of customer expectations force enterprises to acquire
contemporary hardware, resulting in ongoing investments. Beyond that, even with higher costs in the long term, SMEs clearly benefit from the cloud computing pay-per-use price models due to higher liquidity, always up-to-date technology, and increased agility. With higher liquidity, cloud users can invest in their core competencies, enabling more focus on the business. Moreover, being able to avail oneself of an entire ICT-system for a manageable and adjustable price as offered by cloud providers, the entry barrier for SMEs is drastically lowered, which facilitates an easier as well as faster start.

**Vendor lock-in/Data lock-in:** Vendor lock-in is one of the major risks IT decision makers see themselves exposed to, and represents one of the most significant obstacles for reluctant cloud adoption among businesses. A recent study among 500 IT decision makers conducted by Rackspace has revealed that 86% of the respondents consider vendor lock-in as a risk (Bourne, 2012). The reason for vendor lock-in traces back to provider-specific APIs, protocols, standards and tools for cloud computing, which, as opposed to generally interoperable software stacks, are mostly proprietary (Armbrust, et al., 2009). Figure 2 summarizes different forms of vendor-login for IaaS, PaaS and SaaS offerings. Such lock-ins hinder or at least heavily impede cloud customers from migrating their data and software from one provider to another due to high complexity and high costs, making cloud computing users highly dependent on the provider. Joe McKendrick (Forbes) and Thomas Erl (CEO of Arcitura Education) describe the cloud-related lock-in paradigm as "a step backwards from all the work that has been done with approaches such as service oriented architecture" (McKendrick & Erl, 2011). Additionally, Erl currently sees no reason for providers to standardize their platform tools since locking in the users into their environments avoids customer fluctuation and allows some space for markup. According to Erl, an elimination of lock-in effects can only happen if enough customers demand standardization.

<table>
<thead>
<tr>
<th>Type of Lock-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal</td>
<td>Restricted ability to replace with comparable product</td>
</tr>
<tr>
<td>Vertical</td>
<td>Solution restricts choice in other levels of the stack</td>
</tr>
<tr>
<td>Inclined (Diagonal)</td>
<td>Buy other solutions from same vendor, even if not optimal</td>
</tr>
<tr>
<td>Generational</td>
<td>Applies even if no desire to avoid hor/vert/diag lock-in</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Lock-in</th>
<th>Traditional</th>
<th>SaaS</th>
<th>PaaS</th>
<th>IaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infrastructure</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generational</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2 Vendor lock-in and cloud computing (Petri, 2010)

However, there are some recent movements to decrease the vendor lock-in. Those however mostly address vendor lock-in on the infrastructure-level (IaaS): Cloud management provider CliQr Technologies has introduced a product named CloudCenter, which allows "applications to fluidly
move to and between clouds. CloudCenter is an application management platform that enables businesses to quickly and securely deploy, manage and optimize the performance of existing applications on any private, public or hybrid cloud without migration disruption and expense” (CliQr Technologies, 2012). Rackspace’s recently announced OpenStack is “a global collaboration of developers and cloud computing technologists producing the ubiquitous open source cloud computing platform for public and private clouds”.

**Centralized platform for collaborative scenarios, integration hub:** Collaborative organizations, i.e. CNOs and ad-hoc collaborators that convene to co-create typically hit the brick wall in a practical collaborative scenario due to the challenges that global markets pose, including dispersed locations, heterogeneity of the autonomous participants, inadequate ICT-support due to inhomogeneous partner ICT-systems, etc. All too often, collaborative endeavors already fail during the formation stage due to the lack of an efficient and capable common platform for integrated information processing, causing communication problems, inconsistency, or redundancy. In many cases, a dedicated collaboration system needs to be developed and implemented while considering partner-specific ICT-landscape, which leads to a high degree of inefficiency.

With cloud computing, enterprises are given new opportunities to push virtual collaborative alliances to the next level, breaking down barriers and enabling dynamic continuous collaboration that generates globally composed business value. Cloud computing provides a centralized platform, which can be accessed from anywhere, by any (authorized) collaborator, and at any time over the internet. Cloud-based collaboration always allows true collaboration with a single point of contact defined by a collaborative network administrator. The centralized characteristic of cloud platforms allows existing partner-specific applications and software services to communicate with the platform, for instance using a synchronization framework to exchange data. Utilizing the power of a PaaS, collaborative networks can establish an integrated, synchronized, consistent, and ubiquitously accessible SaaS-ecosystem that facilitates collaborative productivity.

**Supporting the volatility of virtual organizations:** "The greatest improvements in the productive powers of labour, and the greater part of the skill, dexterity, and judgment, with which it is anywhere directed, or applied, seem to have been the effects of the division of labor". – Adam Smith

The first sentence of Adam Smith's The Wealth of Nations (1776) alludes to the relationship between productive enhancement and the division of labor, describing one of the major factors, which led to the modern economy as we know today. Our economy is characterized by hyperspecialization with complex global supply chains (Malone, Laubacher, & Johns, 2011). The just-in-time production of the BMW Z4 represents a prime example for intricate supply chains. With global suppliers for over 500 component groups, planning and orchestrating the Z4 supply chain posed a tremendous challenge (Wibbe & Rohde, 2009). When one of the suppliers for sun roofs filed for insolvency during the production phase (Edscha Group), BMW was forced to provide stabilizing support in order to avoid inevitable delays up to six months (Milne, 2009). This drastic measure demonstrates how tightly and interdependently modern collaborative supply chains are organized, and spells out the significant role of each contributing specialist within a collaborative network. The complexity further increases with intangible products (i.e. software), involving data, information, and knowledge that entail more divisibility, dynamic, and pace. Today's ICT allows location-independent as well as decentralized collaboration with immediate response and access to brilliant minds on a global scale. With the rise of new domains, such as cyber physical-systems, traditional supply chains as well as product life cycles of physical products can be extended by services, adding more value, processes, stakeholders, and ultimately complexity to the service-enhanced product, and leading to new forms of collaborative networks that demand more loosely organized schemata. An example for this kind of hybrid collaborative forms that involve collaborative business communities and goal-oriented networks can be found in the solar park industry.

There is an increasing evidence of collaborative networks that involve multi-phase fusion of long-term alliances and short-term goal-oriented virtual organizations that work together during different stages along the value chain. The complementary competences shared among VO participants across all stages eventually fit together the holistic customer experience. Based on the principle of customer excellence, which refers to the constant need for improvement of products and services in order to enthuse customers, global markets can expect a surge of complex value-creating networks aiming at meeting customer wishes (CAS Software AG, 2010) (Basole & Rouse, 2008). Cloud computing
endorses the volatility of virtual enterprises with scalability, flexible cost models, and rapid entry feasibility.
3 Design Goals

This chapter introduces a number of principal design goals (or non-functional requirements) for the GloNet platform. These non-functional requirements come from two fields: the cloud specific nature of the GloNet solutions and the platform-based approach employed by GloNet, which extends and customizes a basic software platform (or framework) with services for collaboratively designing and operating complex products.

3.1 Modularity and extensibility

The GloNet platform provides a comprehensive framework for creating GloNet SaaS applications. Like most frameworks (see for example), the GloNet platform simplifies the creation of new applications and solutions in the following way:

- It defines a basic application architecture that serves as a blueprint for building solutions.
- It implements a number of (reusable) core components. These components provide a good amount of the core functionality common to all applications and solutions that will later be created by the framework. These components can provide infrastructure features (like database support, user interface support,…) but also reusable application features (often referred to as domain or business logic).
- It provides mechanisms to customize existing components towards better solving the requirements of the applications and solutions that are created with the help of the framework.
- It provides mechanisms to extend the functionality of the framework by adding new components to it. Usually this is done by foreseeing and introducing extension mechanisms in certain points of the framework architecture.

As a consequence, the GloNet platform defines some default behavior that can be changed to create a specific application or solution, either by customizing the existing components or by adding new components to suitable points of the platform using the corresponding extension mechanisms.

Since the GloNet platform is targeted towards building SaaS solutions for collaborative scenarios it offers an architectural blueprint for building web- and service enabled cloud solutions. Such solutions typically consist of features to represent and handle data in a flexible way, perform business process related logic with the help of that data and interact with the users through some user interface.

Assuming that the GloNet platform is organized in layers (see section 4.1), this means that the GloNet platform will have to provide a number of mechanisms for configuring and extending its functionality on all layers.

There are two basic ways of achieving this:

- Programmatic extension and customization, i.e. using a module system which allows for adding and replacing modules with specific functionality
- Using a meta data driven approach, i.e. moving from the low-level programmatic approach to a more productive – but limited -- approach through configuration.

In traditional software systems, programmatic extension and customization techniques are often preferred since they generally allow for more flexibility. In cloud scenarios, where a key paradigm is that one consolidated codebase works for all customers, this approach is less suitable.

Instead, software engineers try to implement generic mechanisms on all software layers that interpret customer specific configurations and use these to instantiate data models, business rules and workflows, as well as user interfaces specifically tailored towards the individual users needs. In the literature, such an approach is often called metadata driven (Hicks et al. 1998), because these configurations contain models that define how the user data is structured, displayed and processed.

Sections 4.3 and 4.4 will explain how the GloNet platform addresses these issues.
3.2 Multi-tenancy

Multi-tenancy is the capability of a software system to serve multiple customers or tenants (which in turn comprise multiple users) from a single consolidated software system.

In essence, cloud solutions have to address two potentially conflicting requirements: On one hand they need to leverage the economy of scale principle (see section 2.3) by employing a consolidated architecture that handles all customers uniformly, on the other hand customers demand that the software they use can be tailored to meet their specific requirements and match with their highly-individual business and the processes they work with.

This implies that both data and customizations have to be isolated on a tenant-based level.

Section 4.7 (and to some extend section 4.3) describe how the GloNet platform implements multi-tenancy concepts.

3.3 Scalability and Availability

For any enterprise software, coming up with a scalable software architecture is a major concern. For cloud systems, this is even more critical. The software will be used by thousands of users in parallel, namely the expected average number of concurrent users per customer multiplied by the number of customers having licensed the software.

There are different ways to achieve scalability, see Figure 3. One way is to scale-up the system, i.e. to move the software to more powerful servers (i.e. more processing power, more RAM, more and faster storage,....) when the need arises. This is more or less straightforward from the software architecture’s perspective, however it has serious drawbacks: the older hardware becomes useless, migration to the new hardware might be cumbersome, and scaling down is impossible.

For cloud software, a scale-out strategy is much more suitable. A scale-out strategy means that the workload of the system can be distributed among several servers (often referred to by a farm).

A scaling-out architecture has a number of advantages:

- Instead of using expensive high-end servers, standard and cheap off-the-shelf servers can be used. To increase flexibility and ease maintenance in practice, often a number of virtual machines spread among different physical servers are used in such a scenario. In addition, a monitoring and control unit might be employed to dynamically add or remove these machines to the load-balancing cluster, providing a simple means for ensuring elasticity.
• A scale-out architecture is prepared for higher availability. Since many instances of the same application server are deployed anyway, the architecture has a good amount of redundancy built in.

Section 4.5 describes how the GloNet platform is deployed using a scaling-out paradigm.

3.4 Security and Trust

An important issue in cloud systems is mechanisms for ensuring data security. However, for collaborative solutions like those implemented on top of the GloNet platform, these mechanisms have to be designed in a way that they still allow for data sharing among users that want to collaborate.

In addition to general strategies for application security (see section 4.10) the GloNet platform addresses these aspects using the following mechanisms:

• User, role and permission management (see section 4.8.3)
• Tenant isolation (see section 4.7.1)
• Journaling (see section 4.8.5)

3.5 Network access

Cloud architectures need to be designed for network access from the ground up. This concerns the end user interfaces (web-based UIs) as well as programmatic access (web services).
4 Architecture

4.1 Layered architecture

Most SaaS systems make use of a layered architecture. In the layered architecture (Buschmann et al. 1996), a system is decomposed into several distinct layers or tiers that can be developed, maintained and (often) deployed independently from each other.

- The presentation layer focuses on interacting with the user through a graphical user interface. It displays data and collects user input and commands. In most SaaS systems parts of the presentation layer run in a web browser or on mobile devices (in the form of native clients).

- The application layer or business logic layer provides operations that implement the processes and operations that the software solution provides.

- The data layer encapsulates the storage and provides access to the persistent data of the solution. In most cases this layer makes use of a database management system. Even though non-relational database technologies are getting more widespread with cloud solutions, most SaaS systems are still built using relational database technologies.

4.2 System walkthrough

Figure 4 provides a more detailed view on the software architecture of the GloNet platform. Essentially each layer is implemented using a number of modules. The GloNet platform uses an OSGi run-time (McAffer et al. 2010) to provide an infrastructure for defining, deploying and running independent software modules. Working with that OSGi run-time is structured and simplified using the well-proven Spring framework (Johnson et al. 2005; Walls 2009), which manages the different application modules and their dependencies.

![Figure 4 GloNet system overview](image)

The most important modules of the GloNet platform are the following:

- The data access module encapsulates database specifics. It enforces tenant isolation and implements a powerful permission system, thereby strictly controlling the access to the data stored in the databases. As a clean interface, it provides a database and platform neutral simplified SQL-like language for data access including a number of helper functions to simplify the access of more complex data structures.

- The server core module builds the backbone of the backend implementation of the GloNet platform. It provides mechanisms to register and unregister additional modules, which in turn may provide additional operations. It also is responsible for enforcing a strict security policy.
by providing and verifying security contexts (e.g. based on user and tenant credentials) when executing any kind of operation. Furthermore, it makes use of the data access module to manage generic (extensible) data objects.

- A number of business operation modules provide implementations for the basic services of the GloNet platform. Such operations include infrastructure-related operations (user and tenant management, permission management), but also operations concerning more end user related features (document management, calendar management). New modules can be registered using mechanisms provided by the server core module, creating an extensible platform.

- Any operation from the business operation modules or the server core with a publically defined interface is exposed via a number of external interfaces, namely in-process method calls, RMI and SOAP-style web services. A subset of those is also accessible as REST-style web services. This way, the GloNet platform provides a service-oriented application programming interface (API) suitable for a large variety of use cases and implementation technologies.

- The presentation layer of the GloNet platform provides a framework for creating and presenting a web-based user interface. It is currently implemented using server-side frameworks for AJAX-style user interfaces (Lange 2008), Vaadin and Eclipse RAP. The presentation logic module provides generic user interface functionality like reusable controls, the application frame, configurable list and form views.

4.3 Support for a metadata based configuration

Modern software solutions have to be customizable to a large extend. This is especially the case for software solutions that support implement business processes for companies or organizations. Most businesses are used to being able to tailor their processes to their specific needs. Thus, software systems have to be customized accordingly. Such customization facilities are needed on all layers of the software system:

- User interface: Many users will want to adapt the user interface to match with their corporate branding, i.e. adapt rather simple things like logos, colors and fonts, or to better reflect their internal nomenclature. In many cases users will also like the ability to modify the layout and ordering of the presented information in order to put an emphasis on business critical information and hide less important information.

- Business logic: To optimally support users implementing their daily business using the SaaS solution, a number of basic rules in the business logic layer must be customizable in order to cope with the specific requirements at the customer’s site. A typical example from the CRM domain could be the rules that are used to evaluate the rating or potential of a customer. For one company such a rating might strongly depend on the yearly turnover with that customer, whereas another company might prefer to make that rating dependent on the number of sold licenses (in case of a software company) or the overall value of profitable insurance contracts (in case of an insurance company).

- Data model extensions: In many cases the data models shipped with the SaaS solution are meant to provide a good start for the majority of users. In practice, many users will find that they might want to store additional information in the system or structure the information differently. In these cases extensions of the data model using custom objects or custom fields for each user are needed. Custom data structures require that the underlying data models can be extended appropriately (see also section 4.7.3). In many cases this will also require changes on the user interface level (the newly added custom objects or fields have to be displayed in a suitable place) and on the business logic level (the custom fields may be involved in some of the business rules the user wants to employ).

For GloNet, this approach is shown Figure 5. The platform implements generic mechanisms on all software layers that interpret customer specific configurations and use these to instantiate data models, business rules and workflows, as well as user interfaces specifically tailored towards the individual
users’ needs. Such configurations are usually created with development tools and represented by a \textit{domain specific languages} or XML-based configuration languages.

For example, arbitrary new data objects containing fields of different standard types can be defined and configured using an XML-based data definition. Such a definition allows setting a number of different properties for these fields, e.g. names and labels, lengths for string fields, precision settings for number fields, but also more complex properties such as validation rules to ensure data consistency. On the user interface level, the structure and contents of data forms is also defined by an XML-based UI definition. Using appropriate tool support, software developers, consultants or even customers themselves can rearrange or extend portions of the standard GloNet UI elements, e.g. to show or modify the custom data objects and fields introduced by custom-ized data model definitions. On the business logic level, an example could be the configuration of a generic proxy module for the integration of external web-services, see section 5.1.1).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{Metadata based GloNet platform}
\end{figure}

\section{4.4 Support for dynamic modules}

Generally, adaption and customization through metadata based configuration is a cost-effective way to tailor software systems. However, customization mechanisms only work for cases where the architecture foresees a corresponding configuration mechanism. For a number of scenarios, a more capable and flexible approach is needed, especially for cases where extensions of the functionality of a platform or system are needed.

For this reason, the implementation of the GloNet platform is employing \textit{dynamic modules}, i.e. the platform is decomposed into loosely coupled components that are integrated and wired together using a component infrastructure. For GloNet, this component infrastructure is based on OSGi. OSGi implements a complete and dynamic component model. Applications or components (coming in the form of bundles for deployment) can be installed, started, stopped, updated, and uninstalled into a system or platform without requiring a restart. A registry allows components to detect the addition of new services (provided by new modules), or the removal of services, and adapt accordingly.

The basic functionality of the GloNet platform itself makes heavy use of different OSGI modules (see Figure 6). Additionally it uses OSGi to implement an extensible business logic layer that allows additional operations to be deployed, registered and called into the platform, see section 4.8.
4.5 Deployment architecture

The GloNet platform relies on a sound technical infrastructure. Since runtime requirements like high-availability, scalability and performance are of particular importance for cloud systems like GloNet, it is advisable to employ well-proven hardware and software components when building up the technical architecture. Choosing well-established components helps ensuring a good basic quality but it also helps when deploying and operating the solution – this way, it is much more likely to find dedicated staff with existing know-how in computing centers or service providers.

Figure 7 illustrates the basic infrastructure for the deployment of GloNet. It consists of a farm of standard Linux-based PC-Servers that operate either as Java application servers based on Apache Tomcat or as database servers using MySQL.

All servers make only standard assumptions on the environment they need to operate with. Therefore all servers might also be deployed in the form of standard virtual machines in some modern, scalable virtualization infrastructure. This allows the deployment of the farm in most IaaS offerings available today that allow the deployment of standard virtual machines, such as the Amazon Elastic Cloud or comparable services.
4.6 Technical infrastructure

The GloNet platform employs a number of industry standards and off-the-shelf components which form the technical infrastructure of the GloNet platform. These components are the following:

- Linux-based servers, Apache Tomcat and MySQL, optionally deployed in virtual machines (VMs)
- Spring, incl. Dynamic Module Support (Blueprint), Spring-Security and Spring JDBC-Templates
- OSGi (Equinox)
- EH-Cache (incl. cache sync between servers)
- Eclipse Frameworks: Vaadin, RAP, EMF

These components are standard building blocks of a Java based technology stack. Therefore an instance of the GloNet platform can be deployed on almost any IaaS offering that allows for standard VM images as units of deployment. Adaptions to specific IaaS offerings are only necessary to adapt to the load balancing and VM administration interface of the IaaS offering. Alternatively the GloNet platform can be hosted in almost any computing center that has some expertise in operating Java-based application servers.

4.7 Data layer

The data layer provides the basis for a collaboratively usable Bag Of Information/Knowledge Assets (BOIKA). The section will describe the design of a cloud-enabled, flexible and extensible data layer with advanced mechanisms like a permission management and journaling.

4.7.1 Support for multi-tenancy

A key concern on the data layer level is the isolation of tenant specific data and the mapping of tenant structures to database schemas in such a way that individual customizations of the data models for each tenant are still possible.

The literature discusses different approaches for achieving multi-tenancy on the data layer:

- **One database per tenant**: Each of the tenants is mapped onto a separate physical database for storing the data.
- **One schema per tenant**: Each of the tenants is mapped into separate logical unit, often called schema, within a single physical database.
- **A shared database:** All tenants are stored in the same physical database and schema, but their information is separated using primary keys which are allocated as part of the database design and all the information is kept within the same physical tables.

The first two approaches have advantages with respect to tenancy isolation and data model flexibility. The data associated with each tenant is kept logically (or even physically) isolated from other tenants’ data. Straightforward database techniques can be used for customizing the data models for each tenant individually. The shared-database approach on the other hand has an advantage concerning hosting costs – it is likely that less database server resources are required in this case. However, introducing flexible, tenant specific data models is tricky. There are a number of solutions for this problem (e.g. adding a preset number of data fields that tenants can use flexibly to store custom information (Chong and Carraro 2006), storing tenant specific custom fields in a linked, separate key-value table or employing huge generic data tables with metadata tables to ensure type-safety and pivot tables for optimized querying (Salesforce 2009)).

![Figure 8 Tenant specific database schemas](image)

GloNet employs a one-schema-per-tenant approach. Performance tests have been proven that the GloNet platform will be able to host some 1000 tenants per MySQL database server instance.

### 4.7.2 Mapping multi-tenancy to VO concepts

In a SaaS system, a customer (a company or an organization) is typically associated with one tenant. This approach has the advantage of clearly separating data from different customers. In collaborative scenarios, sharing information between different organizations is a key concept. The GloNet platform therefore introduces a mechanism called *collaboration spaces* for that.

A collaboration space is a shared space in the database that is accessible by an arbitrary number of tenants at the same time. All collaboration spaces are placed in some dedicated tenant. Figure 9 shows such a situation: the database contains two tenant schemas for Customer1 and Customer2 and the dedicated schema for the collaboration spaces. The collaboration spaces schema in turn contains three collaboration spaces, one could be used for a virtual breeding environment (VBE), the other two could be used for two virtual organizations (VO).
To share information (i.e. data object or documents, see section 4.7.3) between tenants and collaboration spaces, replicas of the object are created in each tenant and in each collaboration space. The original object is always referred to as the master object. The platform implementation maintains links between all replicas of the object. The platform can be configured to automatically propagate changes between the replicas; alternatively changes can be propagated in an on-demand mode.

The platform provides a number of operations to create and administer collaboration spaces, some more interesting ones are shown in Table 2.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateCollaborationSpace</td>
<td>Creates a new collaboration space</td>
</tr>
<tr>
<td></td>
<td>Input: name (and additional meta data) describing the collaboration space.</td>
</tr>
<tr>
<td></td>
<td>Output: Unique ID of the new collaboration space</td>
</tr>
<tr>
<td>AddUserToCollaborationSpace</td>
<td>Adds a member to a collaboration space.</td>
</tr>
<tr>
<td></td>
<td>Input: ID of the user, ID of the collaboration space.</td>
</tr>
<tr>
<td>RemoveUserFromCollaborationSpace</td>
<td>Removes a member from a collaboration space.</td>
</tr>
<tr>
<td></td>
<td>Input: ID of the user to be removed.</td>
</tr>
<tr>
<td>ShareObject</td>
<td>Shares an object, thus creating a replica.</td>
</tr>
<tr>
<td></td>
<td>Input: ID of the object, ID of the collaboration space to share the object with; optionally a flag to specify wheater automatic update propagation should be done.</td>
</tr>
<tr>
<td>UnShareObject</td>
<td>Unshares an object.</td>
</tr>
<tr>
<td></td>
<td>Input: ID of the object, ID of the collaboration space.</td>
</tr>
<tr>
<td>PropagateChanges</td>
<td>Propagates changes of an object to another replica.</td>
</tr>
<tr>
<td></td>
<td>Input: ID of the object, ID of the collaboration space.</td>
</tr>
</tbody>
</table>
Technically a collaboration space is implemented as a dedicated data object in the collaboration spaces tenant. Each collaboration space is then modeled using the permission system of the GloNet platform, see section 4.8.3. For each collaboration space, a group is created in the collaboration spaces tenant and every user having access to the collaboration space is added to the group. A dedicated table is used to keep track of the links between the replicas of a shared object.

The concept of collaboration spaces introduces a flexible mechanism for sharing data between companies, VBE and VOs. The infrastructure supports sharing existing data from within tenants or creating dedicated shared objects as well as still restricting information rigorously within one tenant.

4.7.3 Support for generic, customizable data objects

The core task of the data layer in the GloNet platform consists of handling generic data objects. Data objects represent the complex objects of the application domain of a system from a data-centric point of view. Data objects form the so-called domain model or logical data model. In the case of GloNet, BOIK (Bag Of Information and Knowledge Assets) is constructed with the help of these objects.

Since it is unlikely that a fixed, predefined domain model (as well as the implementing physical data model) provided by an instance of the GloNet platform will satisfy all specific requirements imposed by different stakeholders of a GloNet system, the data layer of the GloNet platform is built around a generic, extensible and adaptable data model.

In the GloNet platform, the adaptable data model is implemented by a meta data approach. In addition to the actual user data (or payload data), the platform also stores information that describes the data model to be used to store the user data. This information is called meta information (also: meta data or data schema). The GloNet platform interprets and evaluates this meta information at runtime in order to ensure safe and consistent operations for accessing and manipulating the data.

The basic building blocks for GloNet’s data layer are generic data objects. A generic data type called DataObject represents such objects. The GloNet platform implements operations for defining, accessing and managing such generic objects.

Each DataObject is of some type, e.g. ADDRESS or PRODUCT. This type defines the properties of the corresponding DataObject and the data fields it is composed of. Such a type is represented using the DataObjectDescription data type.

Figure 10 shows the relationship between a DataObject of an example type PRODUCT and its meta object.

![Figure 10 Generic data objects and meta objects](image-url)
Using predefined field types to compose data objects, the GloNet data layer can represent almost arbitrary objects from the application domain.

The data layer provides a set of basic operations to work with generic data objects. Among these are the CRUD (Create, Retrieve, Update, and Delete) operations in Table 3.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating objects</td>
<td></td>
</tr>
<tr>
<td>createObject</td>
<td>Create a new data object.</td>
</tr>
<tr>
<td></td>
<td>Input: the data object type for the new object; Output: a new data object</td>
</tr>
<tr>
<td>duplicateObject</td>
<td>Create a new data object and initialize its value with the values of a given data object</td>
</tr>
<tr>
<td></td>
<td>Input: a data object; Output: a saved duplicate</td>
</tr>
<tr>
<td>Retrieving objects</td>
<td></td>
</tr>
<tr>
<td>getObject</td>
<td>Retrieve a data object with an already known ID (GUID).</td>
</tr>
<tr>
<td></td>
<td>Input: the GUID; Output: the corresponding data object</td>
</tr>
<tr>
<td>query</td>
<td>Query the database using an SQL-like query, returning a list of data objects</td>
</tr>
<tr>
<td></td>
<td>Input: an SQL-like query; Output: a list of data objects matching the query¹</td>
</tr>
<tr>
<td>Saving objects</td>
<td></td>
</tr>
<tr>
<td>saveObject</td>
<td>Save a single data object to the database</td>
</tr>
<tr>
<td></td>
<td>Input: a data object instance</td>
</tr>
<tr>
<td>saveMultipleObjects</td>
<td>Save multiple objects to the database within a single transaction</td>
</tr>
<tr>
<td></td>
<td>Input: a list of data objects</td>
</tr>
<tr>
<td>Deleting objects</td>
<td></td>
</tr>
<tr>
<td>deleteObject</td>
<td>Mark an existing data object as deleted, but do not remove its entry from the database. Deleted data objects are moved to a recycle bin and can be restored if necessary.</td>
</tr>
<tr>
<td></td>
<td>Input: a data object</td>
</tr>
<tr>
<td>purgeObject</td>
<td>Remove the corresponding entry from the database. After purging a data object, it cannot be restored anymore</td>
</tr>
<tr>
<td></td>
<td>Input: a data object</td>
</tr>
</tbody>
</table>

Table 3 Data access operations

Additionally, the data layer needs to provide operations to define the structure of data objects, or more precisely, define the data schema.

¹ The implementation actually returns a data structure called MassQueryResult. A MassQueryResult basically contains a list of rows, where each row corresponds to a selection of fields from a data object matching the query. This structure allows for more efficient queries by retrieving only a subset of the fields of the corresponding data objects.
The GloNet platform provides a number of operations for this purpose.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UpdateDatabaseSchema</td>
<td>Updates the data model (schema) by creating a new data object type for a tenant, or by modifying an existing one. Performs all data-layer specific actions that are needed to support the new data object type.</td>
</tr>
<tr>
<td></td>
<td>Input: XML-based definition of a new or existing data object type.</td>
</tr>
<tr>
<td>DeleteDataObjectType</td>
<td>Delete an existing data object type.</td>
</tr>
</tbody>
</table>

Table 4 Operations for defining the data schema

For the definition of new data object types the data layer of the GloNet platform uses an XML-based specification.

Figure 11 shows a simplified definition of an example data object type PRODUCT which might represent a product data type in the GloNet BOIKA. In the example, a PRODUCT is described by a number of String fields to hold a keyword, provider and customer identifications, a text field for notes and field for a predefined selection of values to represent some life-cycle information.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<dataTypeDefinitionBundle xmlns="http://types.datadefinition.server.open.cas.de">
  <dataTypeDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ObjectTypeDefinition" description="Products" name="PRODUCT" GGUID="B1BE197C211540E8A42636D50BAAC33D">
    <fieldList>
      <field name="KEYWORD" type="STRING" GGUID="C000087209E6447CA98D8A213053A7EF" precision="80" />
      <field name="PROVIDER" type="STRING" GGUID="2A157F65875645F090DF9721B6A6EE92" precision="80" />
      <field name="CUSTOMER" type="STRING" GGUID="00E82B42449C4BEE9AF60E1B1982CB71" precision="80" />
      <field name="NOTES2" type="TEXT" GGUID="78A00672578647F6AC17A4FAE8668B46" />
      <valueListField name="PROJECTPHASE" type="SELECTIONVALUELIST" GGUID="B2660C1543324507B81BFFF1BC37B1A3" cardinality="0.1" />
    </fieldList>
    <!-- Set Up -->
    <value GGUID="810BA6F1F5924D0995C2EECBF4B897B" />
    <!-- Planning -->
    <value GGUID="ADBBC21DF0174D6AE5722CB6AD9C09" />
    <!-- Construction -->
    <value GGUID="95BD988B34AB3C94B47B61C85CF45F3" />
    <!-- Operation -->
    <value GGUID="E5B485E2ACC847F3265DA7B71DE71FA" />
    <!-- Maintenance -->
    <value GGUID="18AF0B6C9424E9F2A75AA7BE95E040" />
  </valueListField>
</dataTypeDefinition>
</dataTypeDefinitionBundle>
```

Figure 11 Example data object definition

Table 5 shows the field types that can be used to define data object types in GloNet.
<table>
<thead>
<tr>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Represents fixed length strings of Unicode characters. Restriction properties can be used to define the maximum length.</td>
</tr>
<tr>
<td>Text</td>
<td>Represents unlimited length Unicode texts.</td>
</tr>
<tr>
<td>Binary</td>
<td>Represents unlimited length byte streams.</td>
</tr>
<tr>
<td>Boolean</td>
<td>Represents Boolean values (true/false)</td>
</tr>
<tr>
<td>Date</td>
<td>Represents time stamps, i.e. a combined date and time value. Restriction properties can be used to limit these fields to either pure date or pure time values.</td>
</tr>
<tr>
<td>Decimal</td>
<td>Represents real numbers. Restriction properties can be used to define the allowed precision.</td>
</tr>
<tr>
<td>Int/Long</td>
<td>Represents integer numbers.</td>
</tr>
<tr>
<td>SelectionValueList</td>
<td>Represents a selection from a predefined list of values. Restriction properties can be used to define the allowed cardinality of the selections.</td>
</tr>
<tr>
<td>Currency</td>
<td>Represents a currency value (wrapper type for Decimal)</td>
</tr>
</tbody>
</table>

*Table 5 Field types in GloNet*

The GloNet platform is designed to work with off-the-shelf relational database management systems such as MySQL or Microsoft SQL server. Therefore both user data and the corresponding meta data have to be saved in database tables.

Using an example data object type ADDRESS, Figure 12 shows how the data layer concepts introduced so far are mapped to database tables.

- **Table TYPES:** Stores a list of data object type names that are known within the tenant specific domain model. In addition to the type name, data object type-specific properties can be defined.

- **Table FIELDS:** Stores the definitions of the fields for the data object types defined within table TYPES. Such a field definition contains at least the following attributes (i.e. table columns):
  - **TYPE_ID:** The ID of the domain object type within table TYPES this field belongs to.
  - **NAME:** The name of the field.
  - **TYPE:** The field's type, see Table 5.
  - Additional Attributes can be used to further describe the field. An example can be PRECISION to specify the for variable-length types like STRING or TEXT or the precision for numbers.

- **Table ADDRESS:** This table stores the user data for the data object type address, i.e. all data objects instances of the data object type ADDRESS are represented as rows in this table.

The GloNet runtime automatically maintains the tables TYPES and FIELDS and updates the structures of the ADDRESS table. In our example, the GloNet runtime evaluates the XML-snippet in Figure 12 upon receiving a UpdateDatabaseSchema operation (see Table 4), populates the tables TYPES and FIELDS and maintains the table ADDRESS with user defined columns STREET, ZIP and TOWN. Additionally it adds the columns INSERT_USER, INSERT_TIMESTAMP and UPDATE_TIMESTAMP to ADDRESS. The latter are required by the GloNet platform to support journaling features (see section 4.8.5). In order to support a permission system, the data object type
ADDRESS needs an accompanying table ADDRESS_RIGHTS in the database that stores access information (see section 4.8.3).

![Diagram of data objects and tables]

**Figure 12 Mapping data objects to database tables**

### 4.7.4 Relations

Relations are a first-level concept in GloNet: any data object can be linked to any other data object. Additionally a number of attributes can be defined for each relation. This concept offers vast possibilities to group data objects together and to make the relations between them visible and evaluable.

Figure 13 shows an example of how data objects can be related to each other:

![Diagram of related data objects]

**Figure 13 Relations between arbitrary data objects**

The appointment data object "CAST-Meeting" is related to an address data object "76131 Karlsruhe" and a project data object named "CAST" at the same time. The "CAST" project data object itself is related to a different appointment named "Presentation".
In order to implement the relation concept within the GloNet platform, a table schema, a relation abstraction and operations to create, retrieve and delete links are needed.

The relations themselves are stored in the database using a dedicated database table named TABLERELATION. Table 6 shows the structure of the TABLERELATION.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUID</td>
<td>GUID</td>
<td>Relation identifier</td>
</tr>
<tr>
<td>GUID1</td>
<td>GUID</td>
<td>Identifier of the left record</td>
</tr>
<tr>
<td>OBJECTTYPE1</td>
<td>String</td>
<td>Data object type of the left record, e. g. &quot;ADDRESS&quot;</td>
</tr>
<tr>
<td>ROLE1</td>
<td>String</td>
<td>Relation role descriptor of the left record</td>
</tr>
<tr>
<td>GUID2</td>
<td>GUID</td>
<td>Identifier of the right record</td>
</tr>
<tr>
<td>OBJECTTYPE2</td>
<td>String</td>
<td>Data object type of the right record, e. g. &quot;PRODUCT&quot;</td>
</tr>
<tr>
<td>ROLE2</td>
<td>String</td>
<td>Relation role descriptor of the right record</td>
</tr>
<tr>
<td>RELATIONNAME</td>
<td>String</td>
<td>Relation attribute identifier, e. g. &quot;Evaluation&quot;</td>
</tr>
<tr>
<td>LINKDIRECTION</td>
<td>String</td>
<td>Direction descriptor, can be &quot;LEFT_TO_RIGHT&quot;, &quot;RIGHT_TO_LEFT&quot; or &quot;BIDIRECTIONAL&quot;. This affects navigability from one data object to another related one.</td>
</tr>
</tbody>
</table>

Table 6 Schema of TABLERELATION

A specific relation -- also called a link -- in the GloNet platform is represented by a LinkObject that encapsulates all the relevant data in Table 6 such as the GUIDs and roles of both related objects, the link direction and optionally a link attribute. There are dedicated operations needed to work with links in the GloNet platform:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveLink</td>
<td>Create one or more links between two different data objects</td>
</tr>
<tr>
<td></td>
<td>Input: list of LinkObjects to save</td>
</tr>
<tr>
<td></td>
<td>Output: list of successfully saved and failed links (e. g. because of insufficient permissions on a data object)</td>
</tr>
<tr>
<td>GetLinks</td>
<td>Retrieve all links matching a given pattern</td>
</tr>
<tr>
<td></td>
<td>Input: LinkObject serving as search pattern</td>
</tr>
<tr>
<td></td>
<td>Output: list of found links</td>
</tr>
<tr>
<td>DeleteLink</td>
<td>Delete one or more given links</td>
</tr>
<tr>
<td></td>
<td>Input: list of GUIDs indicating to the link objects to delete</td>
</tr>
<tr>
<td></td>
<td>Output: list of links the deletion failed for</td>
</tr>
</tbody>
</table>

Table 7 Operations for managing relations between data objects

A common use case for working with links is querying a list of linked data objects. The GloNet SQL-like query syntax offers special predicates for retrieving a list of data objects that are linked to other data objects matching a given criteria. IsLinkedToWhere() is one of these filter expressions. Figure 14 shows an example:
This query retrieves the product name of all data objects of type PRODUCT that are linked to ADDRESS data objects with the following criteria: The ADDRESS data objects' company names start with the prefix "EU" and the link attribute is "Evaluation". Running this query using the query() operation mentioned in section 4.7.3, a fine-grained server-side filtering of relations becomes possible without the need of further client-side actions.

There are also expressions for identifying data objects that are not linked to others and a particular join on linked objects named LINK_JOIN to retrieve the values of both data object types using a single query similar to the example mentioned in Figure 14.

4.7.5 Implementation structure

The data layer in GloNet consists of two sublayers. The upper layer contains so called Data Access Objects (DAO). DAOs provide the interface to the data layer, i.e. all data access and manipulations are routed through DAOs. DAOs play an important role within the data layer of the GloNet platform:

- Since the GloNet platform provides a permission system for data objects, the DAO's implementation is used to enforce permissions and access rights. Since all accesses to data objects have to bypass the DAO, no unauthorized access will be allowed.
- DAOs are responsible for creating journal entries, i.e. each time a data object changes, a corresponding entry will be created. The journal will store information who changed what values when. This ensures traceability of data object changes over the lifetime of an object.
- DAOs ensure data consistency by checking data model constraints and/or automatically adjust values. These checks and operations performed by a DAO are implemented as DAO-plugins, which are called before and after a core CRUD operation executes. Since data object types as well as consistency conditions are domain dependent, new DAO-plugins can be registered and deregistered at a so-called DAO Plugin Manager. A generic DAO plugin can be used to interpret declaratively specified data constraints.

We can see DAOs and DAO-plugins as a layer that adds high-level features to the data access core implementation.
The data access core implementation forms the lower sublayer. This sublayer provides an abstraction from the specific relational database technologies and systems used to store the actual data in. Its main parts are Data Provider Objects.

Data Provider Objects serve different purposes:

- They map data objects to database table constructs.
- They hide database system specifics such as low-level data types and different vendor-specific SQL- and DDL language dialects.
- They create and maintain connections to the database systems.

An important component of the GloNet platform database access layer that is used by Data Providers is an SQL parser that implements a powerful SQL-like query language.

This SQL parser analyses a SQL command and transforms it into the low-level DBMS specific SQL.

### 4.7.6 Query language

An important concept of the GloNet platform database layer is a powerful SQL-like query language. A high-level SQL-like query language provides a number of advantages:

- **Hiding database system specifics:** Even though SQL is defined in a number of standards, every modern database system employs its own variant of the SQL language. Using a high-level platform-specific SQL-like query language adds an abstraction layer on the native database dialects. Additionally, it allows for database specific query optimization.

- **Providing abstractions:** A high-level query language allows for providing additional high-level predicates for features like relations and permissions. These predicates simplify writing complex queries. An example for such a high-level predicate is shown in Figure 14. Additionally, the Data Provider components can create optimized SQL statements for the specific database system used.

- **Security:** Since the Data Provider components parses and translates each query it can ensure that no malicious SQL code can be executed. Additionally, it takes care that the tenant boundaries (see section 4.7.1) are always enforced.

### 4.8 Logic layer

This section describes the design of an extensible business logic layer.

As depicted in Figure 16, the logic layer consists of two sublayers:

- **Interface layer:** the platform provides a mechanism for publishing new operations (or services) as suitable webservices. It supports a number of different communication protocols, namely SOAP, REST, RMI, or a proprietary direct API.

- **Extensible service layer:** new operations (or services) can be added to the platform as OSGi modules.
4.8.1 Interface layer

The interface layer provides a central façade for accessing the backend functionality of the GloNet platform. By default includes operations for interacting with the data layer:

- Creating, reading, updating and deleting data objects (atomic data) (CRUD)
- Querying data
- Executing mass updates

Additionally, it provides a generic mechanism for executing any business operation which is added and registered with the platform.

The façade is implemented by an interface called EIMInterface. Figure 17 depicts this interface and its functions.

All operations provided by the interface are automatically published in the form of externally accessible service interfaces. These external interfaces are available in different communication flavors such as SOAP style web services, Java RMI or Java in-process method calls (see Figure 16).
4.8.2 Business operations

The basic building blocks of a business logic layer are *Business Operations (BO)*. Together, they make up the extensible service sublayer.

Every single feature of the logic layer of the GloNet platform is implemented in the form of business operations. Business operations are realized as objects that implement the Command design pattern. Business operations must be registered at an operation registry in order to be callable. Registering and deregistering of business operations takes place at an OSGi service called `OperationDispatcherService`. Business operations are usually registered during server startup using Spring’s dependency injection mechanism (see section 4.6), but they can also be (de-)registered during runtime.

In contrast to external client code that can only use the operations provided at the `EIMInterface`, business operations run inside the server scope. As a consequence, they have access to server-internal functions and data structures. However, they always run in some well-defined *security context* (see section 4.8.3).

Business operations are invoked using the following `EIMInterface` methods:

- `execute(RequestObject)` and `executeUnauthenticated(RequestObject)` for standard business operations
- `executeManagementOperation(ManagementRequestObject)` for management operations that are usually restricted to platform administrators, e.g. for changing the global data schema.

The `EIMInterface` forwards the request to an operation dispatcher (see Figure 18). This operation dispatcher looks up a matching business operation *command object* in the operation registry. Which business operation should be executed is identified by a subclass of `RequestObject`, e.g. a `DeleteLinkRequest` identifies the `DeleteLink` business operation. Every business operation is invoked by the operation dispatcher with the corresponding *request object* as parameter. Business operation-specific call parameters can be passed from a client to the server as a payload inside the request object. Each business operation creates a *response object* (a subtype of `ResponseObject`), which is passed back to the client via the `EIMInterface`. Analogous to request objects, a response objects stores business operation-specific results, e.g. status codes, computed values or query results.

![Figure 18 Logic layer implementation structure](image-url)
Request and response objects must be serializable, since they are the objects that are send over the network in case of RMI or webservice calls. In the GloNet platform, request and response classes are not implemented by hand but generated based on an XML description.

Figure 19 shows an example definition of a typical business operation:
/**
 * Business operation that retrieves contracts by date.
 */
public class GetContractsByDate
extends BusinessOperation<GetContractsByDateRequest> {

/**
 */
private GetContractsByDateImpl businessLogicImplementation;

/**
 * Executes the business operation GetContractsByDate.
 *
 * @param req The request object of the webservice invocation.
 * @return The response object of the invocation.
 * @throws DataLayerException always possible (no database, wrong client, ...)
 * @see de.cas.open.server.business.BusinessOperation#execute(de.cas.open.server.api.types.RequestObject)
 */
@Override
public ResponseObject execute(final GetContractsByDateRequest req)
throws DataLayerException {

final GetContractsByDateRequest request = (GetContractsByDateRequest) req;

// extract parameter
final XMLGregorianCalendar date = request.getDate();

// invoke implementation
final MassQueryResult result = this.businessLogicImplementation.getContractsByDate(date);

// add results to response object
final GetContractsByDateResponse response = new GetContractsByDateResponse();
response.setContracts(result);

return response;
}

/**
 * Sets the implementation of the business logic 'GetContractsByDate'.
 *
 * @param implementation the 'GetContractsByDate' implementation to set
 */
public void setBusinessLogicImplementation(
    final GetContractsByDateImpl implementation) {
    this.businessLogicImplementation = implementation;
}
}
The corresponding request and response objects (see Figure 20) are declaratively defined as XML data types as meta information for the OSGi module providing the operation. \(^2\) Suitable Java implementations for these objects are generated automatically using the development tools for the GloNet platform (see section 6).

\[\text{Figure 20 Defining request and response objects for a business operation}\]

The business operation must be registered in order for Spring to initialize it. Figure 21 shows an example registration of the META-INF/applicationContext.xml file corresponding to the operation:

\[^2\] Each OSGi module by convention has a folder called META-INF for various meta information used for the configuration and deployment of the module. In our example the request and response object are placed in META-INF/xsd/GenerateContracts.xml
The following sections describe some basic services of the logic layer of the GloNet platform. All of these are implemented in the form of modular business operations.

### 4.8.3 Support for users, roles and permissions

The GloNet platform provides an extensive framework for the authentication and authorization of its use. The most prominent concepts are:

- **Users, groups and resources.** These are called permission owners. Users can authenticate themselves with the platform’s authentication methods. Users can be organized into groups. For each user, the platform maintains some basic attributes. Additionally an arbitrary number of user properties can be associated with each user. The basic user attributes and the user properties are referred to as user profile.

- **Security contexts.** After authenticating a user login, the platform creates a security context. Any operation runs within this security context. The platform provides different classes of security contexts. User contexts are always associated with an authenticated user, privileged contexts are available for performing privileged operations.

- **Permissions.** Most data elements and operations are associated with permissions. Permissions are used for authorizing the access to data or the invocation of operations.

Table 8 contains some operations to work with permission owners, i.e. with users, groups and resources.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetAllPermissionOwners</td>
<td>Retrieves a list of permission owners. You can define which permission owners you want to return by summing the following constants: Users: 1</td>
</tr>
</tbody>
</table>

---

Figure 21 Registering a business operation
A user can be a member of one or more groups. Permissions granted to a group will be inherited to all members. The concepts described below can be configured for individual users as well as for groups.

Figure 22 shows the building blocks of the access control system.

**Data object type permissions** grant access to a data object type (such as contact, document, appointment etc.). Quite often, certain kind of information in an organization shall only be accessible by a small group of people, e.g. tax and income data of employees or customer contracts within a company. Given a platform that models employee data and contracts as data object types, data object type permissions allow people from the HR department to access the employee records, whereas the contracts are only accessible for key account managers and the legal department.

Levels of data object type permissions:

- **No rights** means that a user does not see any records of this data object type. These records are not displayed in any list for users and cannot be searched for.

- **Read** means that a user sees the respective records. The corresponding records are displayed in lists for users with read access and can be found with the search function. Users can also open and read the records. The record cannot be changed by users with read rights. Users with the read right are not allowed to create new records.

---

3 However, it is advisable to configure permissions based on groups because it is easier to maintain, e.g. if a second user obtains the same role as an existing.
Users with the *edit* right can edit corresponding records. A user can change the values in a record or assign rights to a record for other users.

Users with the *edit, delete* rights can edit or delete the record. If a user deletes a record, this record is moved to the recycle bin. This record is not permanently deleted but can be restored.

A user with *full* rights can view, edit, delete and permanently delete records. If a record is deleted in the recycle bin, it is permanently deleted.

**Column access permission** restrict a data object type permission in terms of which fields can be read or written by a group or a user. A typical use case for column access permissions are financial details, such as an employee’s salary. Only people from the HR department shall be granted the permission to change the salary, however the administration must be able to read the value in order to give out the paychecks.

Figure 23 Example table with columns access permissions

This use case can be realized by defining a data object type permission on employee data for all users (based on a group) and restricting the access to *no access*. A second data object type permission now grants the access for the HR department with no limitations on the records’ fields, i.e. the data object type permission is not restricted by any column access permissions. A third data object type permission now grants a access on the employee data to the administration and a column access permission restricts the access on the field salary to *read*.

Levels of column access permissions:

- *No access* means the user can neither read nor edit a field’s content.
- A user with *read* permission can read a field’s content, but is not allowed to change its value.
- A user with *edit* permission can read and change a field’s value.

A column access permission without a data object type permission has no effect. A column access permission cannot grant a higher right than the corresponding data object type permission, i.e. if the data object type permission grants only the right to read objects of a certain type, a column access permission with level *edit* has no effect.

**Direct permissions** (or data object permissions) define a set of users or groups that are allowed to access an individual record. When saving a new record the creator automatically gets full access to the record if no explicit permissions are provided. A direct permission can be combined with a role such as person in charge or participant.

Levels of direct permission:

- *Read* means that a user sees the respective records. The corresponding records are displayed in lists for users with read access and can be found with the search function. Users can also open and read the records. The record cannot be changed by users with read rights.
- Users with the *edit* right can edit corresponding records. A user can change the values in a record or assign rights to a record for other users.
- Users with the *edit, delete* rights can edit or delete the record. If a user deletes a record, this record is moved to the recycle bin. This record is not permanently deleted but can be restored.
- A user with *full* rights can view, edit, delete and permanently delete records. If a record is deleted in the recycle bin, it is permanently deleted.
Direct permissions and data object type permissions interact. A data object type permission restricts the direct permission, i.e. having a direct permission of *edit* and a data object type permission of *read* results in a read permission on the object.

**Foreign edit permissions** enable typical collaboration scenarios. When creating a new record in most cases a user does not want to think of granting permissions. Foreign edit permission work like an indirect right on a record. They grant a permission of a direct permission owner to a set of users. A foreign edit permission can restrict the level of the inherited right, however, via foreign edit permission a user cannot get a higher right than the granting user/group that has the direct permission.

**Figure 24 Example for foreign edit permissions**

Levels of foreign edit permissions:

- **Read** means that a user sees the respective records if the granting permission owner has a direct permission to at least read the record. The corresponding records are displayed in lists for users with read access and can be found with the search function. Users can also open and read the records. The record cannot be changed by users with read rights.

- **Assign role** grants the privilege to add the granting user as a direct permission owner with a certain role (e.g. person in charge).

- Users with the *edit* right can edit corresponding records if the granting permission owner has a direct permission of at least the same level on the record. A user can change the values in a record or assign rights to a record for other users.

- Users with the *edit, delete* rights can edit or delete the record if the granting permission owner has a direct permission of at least the same level on the record. If a user deletes a record, this record is moved to the recycle bin. This record is not permanently deleted but can be restored.

- A user with *full* rights can view, edit, delete and permanently delete records if the granting permission owner has a direct permission of at least the same level on the record. If a record is deleted in the recycle bin, it is permanently deleted.

A foreign edit permission can be deactivated for individual data objects containing sensitive data by setting a *DataObjectForeignEditPermissionRestriction* parameter when saving the object.

A **group permission** is basically a foreign edit permission that is granted between the members of a group. The levels and details are identical to the ones described above.

Individual business operations such as import or export may require a special permission. **Operation permissions** grant the privilege to execute such an operation. Any business operation can require such an operation permission via a configuration entry in the database.
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SaveAndDeleteForeign-EditPermissions</td>
<td>Saves new and changed and deletes existing foreign edit permission</td>
</tr>
<tr>
<td>GetAllForeignEdit-PermissionsOfUser</td>
<td>Returns a list of foreign edit permissions granted by the given user/group.</td>
</tr>
<tr>
<td>GetResultingForeignEdit-PermissionsForAccessor</td>
<td>Returns a list of foreign edit permissions:</td>
</tr>
<tr>
<td></td>
<td>Permissions directly granted from a user to the accessor.</td>
</tr>
<tr>
<td></td>
<td>Permissions directly granted from a group to the accessor. These permissions will be copied for every member of the granting group.</td>
</tr>
<tr>
<td></td>
<td>Permissions directly granted from a user to a super group of the accessor.</td>
</tr>
<tr>
<td></td>
<td>Permissions directly granted from a group to a super group of the accessor. These permissions will be copied for every member of the granting group.</td>
</tr>
<tr>
<td>GetDataObjectType-Permissions</td>
<td>Returns the data object type permissions transitively for a passed record type, the permission owner and all his/her super groups</td>
</tr>
<tr>
<td>SaveDataObjectType-Permissions</td>
<td>Saves data object type permissions.</td>
</tr>
<tr>
<td>GetColumnAccess-Permissions</td>
<td>Returns the column access permissions for an object type and a user and transitively for all groups the user is a member of.</td>
</tr>
<tr>
<td>SaveColumnAccess-Permissions</td>
<td>Saves column access permissions.</td>
</tr>
<tr>
<td>GetPrivilegedOperation-SetPermissions</td>
<td>Returns the operation permissions for a user/group.</td>
</tr>
<tr>
<td>SaveAndDeletePrivileged-OperationSetPermissions</td>
<td>Saves and deletes operation permissions.</td>
</tr>
</tbody>
</table>

Table 9 Operations for managing permissions

4.8.4 Documents

The concept of generic data object types is the core concept to store many different types of records. But GloNet also requires a proper handling of texts and binary documents. The data object type DOCUMENT consist of three subtypes: Note, Binary file and Form letter.

A note is defined as plain text without formatting. Notes are used to share information within a company or between collaborating companies in GloNet.

Because of its plain text nature this content is readable and editable with built-in user interface controls of the GloNet platform. This provides a fast and fail-save access for every user of the GloNet platform.

The content of this type of documents are stored in a data object using the field type Text. This means that it can be easily saved by using the save operation for data objects offered by the persistence layer (see Table 3).

More complex documents like a PDF, a Word document or images are saved as binary file. They can only be viewed or edited by additional programs.

Therefore the GloNet platform only saves the file data to make it for everyone accessible. In this case the data object contains some structured meta data about the document, but not the binary content. The content of the document is saved to a reserved folder on the server. This keeps the database smaller and faster (better cache efficiency).
The meta data of a document is extensible (see section 4.7.3, "UpdateDatabaseSchema"), but the most important fields are already defined. These important fields are:

- **Keyword**: It's the name of the document.
- **Classification**: This field can be used to organize the documents so that they are more easy to find. A typical example in the case of GloNet would be contract, specification or offer.
- **Notes**: This text field can be used to provide more detailed information about this document. This field can take some unstructured information which doesn't fit in another field.

Further fields are internally used by the GloNet platform to keep track of file versions or storage locations.

If several people, even from different companies, work together, a version control mechanism for the document’s content is handy to describe change, avoid loss of contents and to keep specific versions (i.e. General Terms of Condition 2010, 2011, …) for later referral.

Reading and writing a document requires the meta data and the file content. This means the data object is saved by the persistence layer interface (DAO, see section 4.7.5), while the binary content is handled by a special interface called **DocumentStoreManager** to access binary files.

To encapsulate the work with meta and file data, business operations are used (see section 4.8.2). The following two business operations are required to handle documents.

<table>
<thead>
<tr>
<th>Business Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CheckOutFile</strong></td>
<td>Retrieves the file content of a document. Input: a unique identifier for the document and the version number to checkout a specific version of the document. Output: The binary file content</td>
</tr>
<tr>
<td><strong>CheckInFile</strong></td>
<td>Save a data object of type document and the binary file data. Input: the data object containing the meta data of the document, the document content as binary file data, the file type of the document, i.e. docx or pdf. Optionally a flag indicates if a new version of the document has to be created – in this case a comment can be appended to label the new version. Output: Updated data object with corresponding meta data.</td>
</tr>
</tbody>
</table>

The third subtype of documents is **form letter**. A form letter can be used to write the same letter to many companies or customers. This works with placeholders in MS Word documents.

A possible use-case is to write a personally addressed offer to customers.

The GloNet client can be used to assign recipients by using the address records. A plug-in handles the placeholder replacement in MS Word.

The implementation and the usage of a form letter is very similar to a binary file. What's different is the feature to add recipients. This is implemented by using relations to address records (see section 4.7.4).

Searching for specific documents is critical for most use cases in GloNet. Therefore the GloNet platform provides a full-text search on the document contents using the Lucene framework. Documents can thus be queried for by using the generic query-mechanisms on data object,
additionally they are also full-text indexed using the Lucene framework. The Lucene based index is searchable using some dedicated predicates in the SQL-like query language (see section 4.7.6).

### 4.8.5 Journaling

For many use cases it is important to keep track of the changes different users make to the data. This is especially true if companies work together and make changes to the same data objects.

The journaling mechanism keeps track of all changes to any data object stored in the GloNet platform. It is implemented as part of the persistence layer and gets triggered by the `DataObjectDAO` (see Figure 15) so that every change is automatically tracked.

Table 11 shows the data model for the journal used in the GloNet platform.

<table>
<thead>
<tr>
<th>Name Of Journal Field</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update Timestamp</td>
<td>Date, time of change</td>
</tr>
<tr>
<td>User Description</td>
<td>Name of user who did the change</td>
</tr>
<tr>
<td>Table Name</td>
<td>Name of modified data object type</td>
</tr>
<tr>
<td>Table Guid</td>
<td>ID of modified data object</td>
</tr>
<tr>
<td>Field Name</td>
<td>Name of modified field</td>
</tr>
<tr>
<td>Old Field Value</td>
<td>Old value of modified field</td>
</tr>
<tr>
<td>New Field Value</td>
<td>New value of modified field</td>
</tr>
</tbody>
</table>

**Table 11 Data structure for the journal**

Table 12 shows two example journal entries.

- The first entry describes a change at a document with the ID 123 made by Robert Glaser on June 20th. He edited the field `KEYWORD` and changed the content from "Offer Panel" to "Offer for GE Panel".

- The second entry describes a change at a document with the ID 987 made by Bernd Meier on March 3rd. He edited the field `VALID_TO_DATE` and changed the empty field to December 31th.

<table>
<thead>
<tr>
<th>Update Timestamp</th>
<th>User Description</th>
<th>Table Name</th>
<th>Table Guid</th>
<th>Field Name</th>
<th>Old Field Value</th>
<th>New Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-03-03 17:44:35</td>
<td>Bernd Meier</td>
<td>DOCUMENT</td>
<td>987</td>
<td>VALID_TO_DATE</td>
<td>2012-12-31 00:00:00</td>
<td>2012-12-31 00:00:00</td>
</tr>
</tbody>
</table>

**Table 12 Example journal entries**

Using the information logged in the journal, changes can even be rolled back.

Table 13 shows the operation that is used to query the journal for a specific data object.

<table>
<thead>
<tr>
<th>Business Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetJournal</td>
<td>Retrieves the change history of a data object. Input: the unique identifier for the data object whose journal should be retrieved and the data object type of the object, i.e. DOCUMENT.</td>
</tr>
</tbody>
</table>
4.9 Presentation layer

Like the GloNet platform itself, the presentation layer or user interface is build using a framework based approach. The presentation layer defines a blueprint for building user interfaces and provides a number of predefined components.

A preliminary screenshot of a GloNet solution illustrates these concepts, see Figure 25.

- The application frame (which also defines the basic layout)
- A navigation area (which provides access to the different modules used in a GloNet systems)
- A customizable toolbar where the user can execute a number of commands and operations suitable for the current content displayed in the content area.
- A content area that (depending on the UI workflow) provides a list view with advanced filter and query capabilities for browsing through the GloNet data space or a form view to interact with a specific data object stored in the GloNet platform.

The implementation structure of the GloNet user interface is shown in Figure 26. It heavily relies on the frameworks Eclipse RAP and Vaadin.
A number of the building blocks of the GloNet user interface can be customized and extended. To allow a flexible extension and customization of the user interfaces of a GloNet solution, GloNet provides a mechanism for declarative user interfaces. An XML-based language is used to describe some portions of the UI, e.g., how forms are composed from a number of basic user interface elements (controls) and a number of custom controls. Implementations for additional custom controls may be provided as OSGi components.

Figure 27 illustrates this concept: a user interface portion is defined using the XML-based UI description language. The presentation layer runtime creates and manages the user interface elements necessary for rendering the UI using a pool of predefined and custom controls. It also implements an event handling mechanism. Users interacting with the UI create events that are interpreted by the presentation layer runtime. The presentation layer runtime interacts with the business layer, retrieves or writes back data and updates the UI elements accordingly.
4.10 Security mechanisms

Security and safe operation of the GloNet platform is a very important topic. Cloud systems like GloNet might be seen as an attractive prey to attackers because they operate on the data of many customers and organizations. In general, the literature provides a huge set of methods and tools to help building secure web applications – the OWASP project for example provides a good starting point (OWASP 2012).

With respect to GloNet systems, two things are of particular importance

- **Data protection.** Users should *never* be able to access data from other users unless the latter chose to provide them with some access. Isolation of user and tenant data is a concept, which should be embedded deeply into the platform architecture. In the GloNet platform for example, any operation is strictly associated with a security context, which is always evaluated and enforced when eventually accessing tenant or user data from the database. Details can be found in section 4.7.

- **Safe data access paths crossing subsystem boundaries.** It has been a good practice to use subsystem boundaries also as a bulkhead to contain the impact of security flaws. Each subsystem interface should therefore ensure that it only allows sane, flawless interactions. An example might be a routine in the data layer that checks all incoming SQL expressions for syntactic and semantic sanity to avoid SQL injection. Similar measures should be designed for service interfaces and for AJAX-style user interfaces when validating user input.

Figure 28 maps these concepts onto the GloNet system architecture from Chapter 4.

![Figure 28 Security mechanisms](image-url)
5 Integration mechanisms

Integration with external components, services or systems is a must for distributed/collaborative environments. The GloNet platform architecture provides different integration facilities.

5.1 Service-API

For some scenarios, the GloNet platform has to support the integration of externally deployed services. These services are independently developed and they are deployed somewhere in the cloud (e.g. on Amazon’s elastic cloud or on Google’s app engine or simply as some web application operated by a GloNet partner). Therefore they share no infrastructure with the GloNet platform.

Service integration requires more than just calling a service -- it requires integration into the surrounding application or platform logic. Since external services can have arbitrary interfaces (syntax, protocol, semantics, user interfaces), it is necessary to map GloNet platform concepts onto those of the external services. In order to integrate an external service, the GloNet platform provides two different mechanisms. The main difference between these two mechanisms is the place where the mapping between the GloNet platform and the external service is done. From the GloNet platform's viewpoint, the two mechanisms are:

1. A proxy-based approach where each external service has a proxy within the GloNet platform that performs the mapping between the GloNet platform and the external service. A proxy is implemented as a plugin to the GloNet platform, i.e. the mapping is done within the context of the GloNet platform, leaving the external service unaware of the GloNet platform. Since in most cases these plugins will need to provide glue code to implement both data and protocol mediation, the codebase of the GloNet platform must be extended. Thus, this approach is a rather heavyweight integration. However, a seamless integration into GloNet platform-based applications is possible, since the proxy runs within the same context as the GloNet platform core. It is even possible to add new user interface components that can be used to control and manage the external service (e.g. service specific data input forms).

2. A mashup-based approach where an external service is aware of the GloNet platform, i.e. the external service uses the external webservice interfaces (REST, SOAP) of the GloNet platform to push/pull data to/from the platform. Data and protocol mediation is performed in the external service. The GloNet platform provides a thin integration interface that exposes some basic state of the current application using RESTful services. User interface composition is done within the end user's browser by rendering the external service's user interface into dedicated frames within the applications UI (i.e. the external service is responsible for providing its own user interface). Since no additions to the GloNet platform's code base (e.g. proxy implementations) as well as data and protocol mediation is necessary within the GloNet platform, this is a rather lightweight integration.

In the following sections, we describe the two approaches in a more detailed way.

5.1.1 Proxy-based integration

In general, the mechanism for a proxy-based external service integration works the following way (see Figure 29):

- An event in the GloNet platform triggers the invocation of an external service. For example, such an event can be the execution of a command in the GloNet user interface or a condition (business rule) in the GloNet data model.
- The GloNet platform consults a service registry to identify the external service to be called and associates a suitable service proxy plugin with it.
- The service plugin collects and assembles the data needed as input for the external service. Usually it also transforms the data into the form required by the external services (data mediation).
• It then invokes the services using a suitable web service protocol (SOAP, REST). The actual invocation can either be supported by generic invocation components provided by the GloNet platform, or it can be executed by the plugin itself. The latter also allows for complex invocation protocols.

• The external service executes. Upon termination, the service plugin receives the output of the service execution, transforms it back into a form suitable for the GloNet platform and its components.

• The results of the service invocation can then be passed to the GloNet user interface or used to update the GloNet data model.

![Figure 29 External service integration](image)

The invocation of an external service can be synchronous or asynchronous. While services that provide data or computations which should be displayed within an application are typically invoked synchronously, there might also be external services that run for a long time (minutes to days in case that the service triggers workflows that involve human interaction). The latter type of services are executed asynchronously. After the long-running service has been executed, a special callback servlet within the GloNet platform must be notified by the external service. This notification contains a transaction ID that identifies the context of original invocation (e.g. the user that triggered the invocation) as well as a one-time key that can be used by the external service in order to authenticate at the GloNet platform's security system.

For specific scenarios, generic plugins can be implemented. These plugins will then make a number of assumptions on the external services, i.e. depending on the scenario a service will have to implement a specific interface in order to be integrated. A good example might be services that provide sensor or profiling data for pieces of equipment deployed in a GloNet product. The generic plugins can be configured in a declarative way, which allows an integration of the external service without extending the GloNet platform's code base with the service-specific plugin implementation.

5.1.2 Mashup-based integration

An even more generic service integration is possible for services that provide their own user interface (note that these are not web services in the strict sense). Such services can be integrated into a GloNet system by embedding a frame into the GloNet user interface. To exchange data with the GloNet platform, the webservice API provided by the GloNet platform can be used (see section 5.1). In order to interact with the GloNet application, basic user interface context information is exposed to external services by a small RESTful interface. Since the user interface composition is done within the browser and not using the GloNet platform's UI framework capabilities, this compares to concepts known from *mashups* (Hoyer 2008).
In general, the mechanism for a mashup-based external service integration works the following way (see Figure 30):

- An embedded browser widget (typically implemented using an IFrame) is added to the form. This browser widget displays the user interface of the external service (red rectangle in Figure 30). This widget together with metadata that is needed to configure some basic integration properties (e.g. the external service's base URL) are part of the declarative form specification (see section 4.9).

- When rendering the form, the presentation layer sets the external service URL (communication channel 1 in Figure 30) together with some basic context information (see below). This leads to a browser-side request to the external service (communication channel 2). Context information is passed using HTTP GET parameters.

- The external service can query user interface context information for the user's session, like the currently open object data object or the current selection of a known table (communication channel 3).

- Triggered by user interaction with the external service UI, the external service can interact with the webservice interfaces (REST, SOAP) of the GloNet platform (communication channel 4), as well as querying the current context information (communication channel 3).

Data that is needed by the external service to access the current user's session can be passed as HTTP GET parameters encoded within the URL that is set to the browser widget. Conceptually, the following data will be passed to the external service using the URL:

- A `sessionID` that identifies the user's session. This session ID must be passed to the GloNet platform when accessing context information.

- A `formID` that uniquely identifies the form instance in which the external service UI is embedded.

- A set of `tableIDs` for identifying tables within the form with a given `formID`. These tableIDs can later be used by the external service to query the selection state of the tables. Which tables the external service is interested in is determined by the metadata associated with browser widget.
• An action attribute that exposes important states changes in the lifecycle of the form. If the external service is interested in state changes (specified by a flag in the browser widget's metadata), the URL is set multiple times by the GloNet presentation layer. It is up to the external service to react on these state changes (rendering a new UI, updating its UI, ...). Possible state changes exposed by the GloNet presentation layer to the external service are:

  o OPEN: An existing data object has been opened within the form, i.e. its values are currently displayed in the form.
  o NEW: A new (empty) data object has been created and is displayed in the form.
  o SAVE: The currently displayed and edited data object has been saved.
  o CLOSE: The form has been closed, i.e. the UI of the external service is not displayed anymore.
  o SELECTION_CHANGED: The selection in one of the observed tables has been changed.
  o REFRESH: The content of the form has been refreshed. This event can be used by the external service to update its UI, too.

Using the sessionID, formID and tableID, the external service can query the GloNet application for basic context information. Such context information can be:

• The data object that is currently shown in the form with the given formID to the user with session sessionID.

• The currently selected objects in the table that is uniquely identified by tableID, formID and sessionID. This information can be used by the external service to display selection-specific data.

The objects provided by the presentation layer as context information do not necessarily contain all fields as they are returned by the REST/SOAP webservices provided by the business logic layer. Only the values that are displayed are part of the objects. However, it is guaranteed that the objects contain at least an object type specification together with a GUID (if available). This can be used to query the full objects in case the external service is interested in more detailed information.

The presentation layer optionally can provide a REST service that triggers a full refresh of the form with a given sessionID and formID. This enables the external service to update data stored in the GloNet platform and display these changes immediately to the user. However, such a service requires the support of server-side-push by the UI framework that is used to implement the presentation layer, since it is necessary to contact the browser and update its content without a user-initiated browser request.

5.2 OpenID

Scenarios that involve service integration do often require end-users to authenticate against multiple service providers, obliging the user to remember and enter different credentials for each provider. This fact does not only have a significantly negative impact on the usability. Since no identity information is shared among the involved services, it is impossible for the service providers to make collaborative access decisions based on the user's identity.

To circumvent these problems, the GloNet platform supports the OpenID (OpenID Foundation 2012) for decentralized authentication and identity management. Using OpenID, the platform is able to provide single sign-on (SSO) service integration, which enhances the usability of applications built on top of the platform. Furthermore, OpenID enables the sharing of identity information among integrated services.
The OpenID functionality is deeply integrated with the GloNet platform architecture. The major integration points are the authentication façade and the platform’s user management.

The authentication façade (see Figure 31) is a subsystem of the interface layer which is responsible for authenticating any calls to the platform. Depending on the type of authentication used in the call, the façade selects the suitable authentication provider to perform the actual authentication of the call. The GloNet platform features three authentication providers for OpenID, username/password and SSO Tokens. The authentication façade is modularly expandable, allowing more authentication providers to be added in the future if need be. Upon successful authentication by at least one of the authentication providers, the call is passed on to the Authorization layer where access decisions are made. Otherwise it is rejected.

The nature of OpenID requires linking the user’s OpenID identity to a platform user account. Mechanisms that implement this linking are provided by the platform’s user management, which provides logic and user interface components to establish, query, manage, and clear links between OpenID identities and platform user accounts.

The OpenID authentication provider utilizes the functionality of the user management after the user’s identity has been successfully verified by an OpenID provider to identify the platform user account that is linked to the verified identity for which it issues an SSO Token. This SSO token can be to authenticate subsequent calls, relieving the user from going through the OpenID protocol flow again and again.

5.3 Synchronization framework

The platform has a powerful synchronization framework that allows synchronizing data with external systems such as on premise ERP software. With data we mean individual records as well as relations between those records. As the software tracks the state of records and their relations it works highly efficient, it only transfers data that has really changed. At the beginning of a synchronization session the framework obtains the state from each of the two synchronized systems and compares them with the known state of the last session.
Figure 32 Synchronization philosophy
5.3.1 The architecture on a bird's eye view level

The architecture on a bird's eye view level is the following:

![Diagram of synchronization framework](image)

The **ConnectionBridge** module controls the communication between the sync framework and a system to be synchronized. To ensure an independent, reusable implementation the bridge encapsulates the used technology of communication between the system and the synchronization framework. The bridge has no domain model-based dependencies which allows using the same implementation for different systems that support the implemented technology. For example, a MySQL Client Bridge can be used for all systems that support a direct connection to the underlying MySQL database. The differences in the domain model structure of various systems are resolved using the mapping and link definitions.

The **ConnectionBridge** interface defines a small set of methods whose implementation is needed to fulfill the requirements of the synchronization process. To support new systems, this interface needs to be implemented but the synchronization process itself need not be changed. This keeps the cost of supporting multiple external Systems low.

The **Sync Framework** itself manages the workflow of the entire synchronization session and manages the communication with the **SyncDao** and the **ConnectionBridges**. It is stateful throughout the sync session. It does not store any data between the sessions. It also contains the logic of the synchronization. This includes the ability to determine the states of the records to synchronize, the creation of the operations based on the record pair state and the strategy. The delegation of requests for meta information and configurations for the synchronization session to the **SyncDao** is also done by the **SyncHandler**.

The **SyncDAO** holds the following state about the synchronization for each system:

- pair information,
- set of synchronized records,
- last sync identifier for each record
- a configuration
5.3.2 Synchronization concepts

The pair information

The synchronization is based on the notion of pairs: a pair is the association of two records, one from the GloNet platform and the other from the external or 3rd party system, that both represent the same data.

This association is often based on the ID (GUID) of the record that is unique (at least) in its system.

If a record is synchronized from the GloNet System to an external system such as an ERP system (i.e. created there), the existing and the newly created records are associated with this pair association. This enables the system to track consecutive changes and apply them on the pair of the records to change.

Note: the pair association is a sort of identity association, and is more specific than a "has the same data" relation. i.e., if there are 2 address records, both created manually in the GloNet platform and the ERP system, they are not recognized as pairs, even if they have the same data. Hence, it is always true that one side of a pair was created manually and the other was created by the synchronization module.

The Last Sync ID/Timestamp

The Last Sync ID denotes a point of time in the past, when a record was last synchronized successfully. It is necessary to track changes only after this last sync id. (Changes before these are not interesting as they have already been synchronized.) Additionally a sync session includes all previously failed records.

Changes on a time bar:

- Last synchronisation
- Record that failed in last sync session
- Record successfully synchronized in last session
- New record
- Records processed in next sync session

*Figure 34 Determining records to synchronize*

Often the value of an update timestamp field at the time of the synchronization is used to set the last sync timestamp value. Note however, if the system does not support update timestamp, hashes on the full representation of the record or incremented counters of a journal can be used instead.

In the following diagrams, we call the state of a record in-sync, when it fully corresponds to its pair.

Based on the Last Sync ID and the update timestamp of a record, we can determine whether the record was changed since the last synchronization, or is still unchanged.

Deriving sync strategies

The state of a pair is a combination of the states of its constituent records, e.g. unchanged-unchanged, changed-deleted …

Note: a record that is new does not have a pair (or its pseudo pair has the non-existent state).

A pair is in conflict, if both of its records have a changed or missing/deleted state, e.g. changed-deleted.
The **sync strategy** associates a pair state with an **operation**. The operation is executed on a pair, with the ultimate goal to synchronize the records of the pair. A simple operation acts only on one of the records (e.g. update client; delete client), whereas a merge operation acts on both records (e.g. update client AND update server; create client AND update server).

There are many conceivable strategies: such as bidirectional synchronization, unidirectional synchronization (changes of the slave are lost / preserved).

The strategy resolves the conflicts by associating an operation to each conflict pair state.

E.g., a client wins strategy could be as follows:

- **Changed client** – **changed server** -> update server (thereby overwriting the changes on the server)
- **Changed client** – **deleted server** -> create server (recreating the deleted record on the server)

**Sync Views**

To limit the number of synchronized records per session, it is possible to configure views (represented by a *SyncViewDefinition*). These views contain a record type, e.g. address and an optional filter which can specify a condition the synchronized records must achieve. For example, a filter on a connection which is working with an SQL-Database could be a simple *where*-condition like: "employee = 1". In this case a sync session using this view would only process address records which are flagged as employees.

**Concurrency**

The records that are synchronized are not locked in either system, i.e. concurrent changes during a synchronization session are possible.

Locking the records, i.e. preventing changes in the system is not a valid option as the system should continue to work just as it would without synchronization. Users of the system or other entities should be able to create new records and modify existing ones during synchronization.

As concurrent modifications cannot be prevented, we need to recognize and handle them.

It is important to note, that the synchronization of each record is handled separately: i.e., if a pair is synchronized, and after that, but during the synchronization session, a concurrent change happens, it is no problem as this change is going to be handled in the next synchronization session. i.e., the last last sync timestamp is stored separately for each record.
However, if a concurrent modification happens after the start of the synchronization session but before the synchronization operation for the pair can be executed, the synchronization of the pair is aborted. (Note: the synchronization of other, independent pairs is not affected by this).

As the last sync timestamps are not updated, in the next synchronization session the changes before this session are going to be considered, as well.

Note: If not being able to lock any of the systems, let alone both of the systems, an automatic merge operation (modifying both sides) is not possible. Just like in version control systems, a manual merge is needed in case of concurrent modifications, that iterates as long as there are no concurrent modifications any more.

---

**Figure 36 Conflict free synchronization situation**

---

**Figure 37 Synchronization situation with conflicts**
Sequence of actions during the synchronization

The sequence of the actions in a synchronization session is as follows:

1. **Init with view- and link definitions**
   The ConnectionBridge is initialized and gets the view definitions and link definitions which must be cached to be able to use these for further calls.

2. **Get change set**
   The sync framework has to retrieve the current change set for the systems. Because the change set could be too huge to send it at once, an iterator is used to retrieve the set.

3. **Change set with record and link info.**
   The bridge must return all records and links information which are contained in any of the definitions. The records must match the record types and filters given in the view definitions. The links must match to one of the link definitions.

4. **Get data for IDs**
   After resolving all states the sync framework can determine which operation should be executed per record. Some Operations needs the data of the record so this step has to return the data for all given record IDs.

5. **Request data for ids**
   Return the data for the requested record IDs.

6. **Operations (CUD)**
   The operations have to be executed now.

7. **Results of the operations:**
   After executing the operations, the bridge sends the result (success, failure) and the newly created IDs to the server.

   The sync framework can update the pair information (create a new pair for the newly created record) and increment the last sync timestamps for the successfully synchronized records.

![Figure 38 Synchronization sequence](image-url)
Mapping of fields and types

To convert the domain model of the client system in the server side model and the other way round, the sync framework uses a XML-based mapping definition language. In this way the mapping handles the domain model specific structures.

Figure 39: Mapping of fields and types to be synchronized

Below is a little example of such a XML-based record mapping which maps the client record type "MyAddress" and the two fields "MyStreet" and "MyCity" to GloNet server based record type ADDRESS.

```xml
<mapping recordtypeid="MyAddress">
  <serverobjecttype name="ADDRESS"/>
  <clienttoservermapping>
    <serverprimarykeys>
      <serverfield name="MyAddressPrimaryKey"/>
    </serverprimarykeys>
  </clienttoservermapping>
  <servertoclientmapping>
    <clientprimarykeys>
      <clientfield name="GGUID"/>
    </clientprimarykeys>
  </servertoclientmapping>

  <fieldmapping direction="both">
    <clientfields>
      <clientfield name="MyStreet"/>
    </clientfields>
    <serverfields>
      <serverfield name="STREET_1"/>
    </serverfields>
  </fieldmapping>

  <fieldmapping direction="both">
    <clientfields>
      <clientfield name="MyCity"/>
    </clientfields>
    <serverfields>
      <serverfield name="TOWN_1"/>
    </serverfields>
  </fieldmapping>
</mapping>
```
Synchronization of relations

Each system could have its own structure of relations or links that are semantically equivalent and should be synchronized.

The structure of links contains:
- the domain model of records that are linked
- the definition of a link (e.g., foreign key vs. separate relation table)
- the order of record creation (e.g., when a foreign key is a mandatory field in a record)

To synchronize these links each ConnectionBridge implementation has to be able to find and create the links in its own way. To allow this, the sync framework will only work with the semantic of a link and its source and target records. The structure of these links is encapsulated and used in the ConnectionBridge only.

The Link-Container is the instances of links in the sync framework and contains:
- Source id: The id of the record the link is outgoing from.
- Target id: The id of the record the link comes in.
- Link definition name: The name that defines the semantically sense of the link instance and maps these instance to the corresponding link definition.

The LinkDefinition is the system-specific link structure and contains:
- Name: Describes the link semantic and used to identify equivalent link definitions on the different systems. Used to identify which link instances should be handled with this definition.
- Definition: Contains all necessary information to retrieve or create links on the specific system, e.g., a table name and field name the foreign key has to be inserted to link the records.

The Order of record creations

In some system it is possible, that the link structure determines the order the records have to be created. This happens if two records are linked by a foreign key field that is mandatory. So the record providing the id for the foreign key field must be created before the other record. The sync framework will sort the operations before passing them to the ConnectionBridge, so that the ConnectionBridge does not need to concentrate on the order of creating the records. To mark links that have a relevant order of creation, a link definition can contain a constraint that shows which record should be created first. The possible constraints are SOURCE FIRST, TARGET FIRST or NONE.

A synchronized system's data model may require a set of records to be created at once. An example would be an order together with its positions. Thus, these kinds of links are sent to the ConnectionBridge as part of the CreateOperation, which creates the records in the target system having this data model constraint.
6 GloNet development tools

In order to customize and extend the GloNet platform and to implement applications and solutions based on the GloNet platform, a development environment as needed.

The GloNet development environment is built by using and extending the Eclipse development environment.

- Eclipse already provides good tools support to write Java and OSGI based modules which make up for the core of the GloNet platform.

- A modular build environment based on Maven will simplify assembling applications from the components that platform provides.

- Eclipse can be easily extended with additional tools. This capability will be used to provide custom editors for the configuration languages and meta data descriptions employed by the GloNet platform, see Figure 5. These editors can either be text based or more intuitive graphical editors.

- In later phases of the development of the GloNet platform, graphical editors will be developed at least for specifying GloNet data models and for designing GloNet user interfaces.
7 Conclusion

The GloNet platform provides a comprehensive framework for creating applications for the collaborative design and operation of complex service enhanced products.

GloNet applications and services are designed to make use of the cloud. As a consequence, the GloNet platform is designed and implemented as a framework that helps creating the GloNet applications and services as SaaS offerings.

The Glonet platform provides both an architecture blueprint for such applications as well as basic components and services as building blocks of this architecture.

Specifically, the GloNet platform provides the following features and components:

- A modular, extensible platform architecture. An application build with the GloNet platform can customize and extend a basic application skeleton on all layers: on the data layer, on the logic layer and on the user interface layer.

- Depending on the needs, two basic mechanisms are available: customization through configuration based on metadata or programmatic extension by solution specific plugins or modules.

- A number of basic components provide important basic services (including user interfaces) for collaborative applications, especially with respect for building up a bag of information and knowledge assets:
  - A data layer and corresponding operations/services with advanced features like advanced query capabilities, a user and permission system, journaling and document support.
  - An infrastructure for sharing information using collaboration spaces.
  - Additional modules for typical customer relationship management and groupware related functionalities, e.g. address management, team calendar management, task and project management.

- Additionally, the platform provides mechanisms for the integration of external systems and services.
  - A plugin-based mechanism for integrating external web services. This mechanism can be used, if existing service implementations should be loosely integrated into a GloNet solution – whenever a tight and seamless integration using OSGi modules (for example in the form of business operations deployed to the platform) is not feasible.
  - A framework for data synchronization. This mechanism is well suited for the integration of mass data that is already available in legacy systems operated by some of the partners in a GloNet collaboration scenario. Typically examples include ERP systems or system monitoring tools that are used to monitor sensors during the operation of GloNet product.
8 REFERENCES


9 ANNEX

9.1 EIMInterface, the platform’s main façade

```java
package de.cas;

import java.util.List;
import de.cas.open.server.api.types.DataObject;
import de.cas.open.server.api.types.DataObjectDescription;
import de.cas.open.server.api.types.IndexedGuidEIMExceptionContainer;
import de.cas.open.server.api.types.ManagementRequestObject;
import de.cas.open.server.api.types.ManagementResponseObject;
import de.cas.open.server.api.types.MassQueryResult;
import de.cas.open.server.api.types.MassUpdateData;
import de.cas.open.server.api.types.RequestObject;
import de.cas.open.server.api.types.ResponseObject;

/** Facade to access the GloNet service layer. */
public interface EIMInterface {

    /** This method is used to execute business logic on the server. */
    public ResponseObject execute(RequestObject requestObject);

    /** Same as execute, but it’s usable for business operations which don’t
     * need a authentication. */
    public ResponseObject executeUnauthenticated(RequestObject requestObject);

    /** This method creates a new data object. */
    public DataObject createObject(final String objectType);

    /** Creates a new (saved) DataObject from a template. */
    public DataObject createObjectFromTemplate(final String objectType,
                                                  final String templateGUID);

    /** This method retrieves a data object. */
    public DataObject getObject(final String typeName, final String gguid);

    /** This method saves a generic DataObject. Only after this method,
     * a data object is persistent. */
    public DataObject saveAndReturnObject(final DataObject dataObject);

    /** Saves multiple objects within an transaction. */
    public List<IndexedGuidEIMExceptionContainer> saveMultipleObjects(
                                                                 final MassUpdateData data);

    /** This method deletes a data object. After this method, the object is
     * deleted but still available in the recycle bin. */
    public void deleteObject(final String typeName, final String gguid);

    /** This method undeletes a data object which was previously tagged as
     * deleted (placed in the recycle bin). */
    public void undeleteObject(final String typeName, final String gguid);

    /** This method purges a data object. It will be removed unrecoverably. */
    public void purgeObject(final String typeName, final String gguid);

    /** Executes a query (only SELECT) and returns a MassQueryResult object
     * containing an entry for each returned row and the types of the fields
     * in the rows. */
    public MassQueryResult query(final String query, final long firstRecord,
                                 final long recordCount);

    /** Returns the number of records a call of the method query() would return.
     * This is useful for paging. */
    public int getQueryRecordCount(final String query);

    /** This method returns meta information for a DataObject type.
     * It contains all field names and the type of the fields. */
    public DataObjectDescription getObjectDescription(final String typeName);
}
```
/** Returns a list with all available DataObject types (i.e. Appointment, * Document, Contact, ...). */ * /
public List<String> getAvailableObjectNames();

/** Duplicates the <code>objectToDuplicate</code> including all permissions, * tags and links. */ * /
public abstract DataObject duplicateObject(String objectType, String gguid, boolean copyLinks);

/** This method is used to execute management operations on the server. */ * /
public ManagementResponseObject executeManagementOperation(
    ManagementRequestObject requestObject);

9.2 Examples for using the GloNet webservice API

This section provides some example code for using the service interface layer of the GloNet platform. It hides the technicalities of the SOAP style webservice infrastructure behind a client side façade that also implements the EIMInterface (see 4.8.1).

The examples create and save a data object of type APPOINTMENT and retrieve a document. The code for retrieving a document also demonstrates the use of the execute-METHOD used for calling business operations (see 4.8.2) defined by the GloNet platform.

Package de.cas;
import javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;
import de.cas.eim.PropertyConfiguredTestCase;
import de.cas.open.server.api.exceptions.BusinessException;
import de.cas.open.server.api.exceptions.DataLayerException;
import de.cas.open.server.api.types.DataObject;
import de.cas.open.server.appointments.constants.APPOINTMENT;
import de.cas.open.server.documents.constants.DOCUMENT;
import de.cas.open.server.documents.types.CheckInFileRequest;
import de.cas.open.server.documents.types.CheckInFileResponse;
import de.cas.open.server.documents.types.CheckOutFileRequest;
import de.cas.open.server.documents.types.CheckOutFileResponse;

public class GloNetTests extends PropertyConfiguredTestCase {

    public void createNewAppointmentTest() throws DataLayerException, DatatypeConfigurationException {
        // create new appointment object (unsaved)
        final DataObject unsavedApp = eimInterface.createObject(APPOINTMENT.value());

        // set data to new appointment
        unsavedApp.setString(APPOINTMENT.KEYWORD, "Discussion GloNet");
        unsavedApp.setBoolean(APPOINTMENT.WHOLEDAY, false);

        final XMLGregorianCalendar startDt = DatatypeFactory.newInstance().newXMLGregorianCalendar(2009, 12, 10, 9, 30, 0, 0, 0);
        final XMLGregorianCalendar endDt = DatatypeFactory.newInstance().newXMLGregorianCalendar(2009, 12, 10, 10, 30, 0, 0, 0);
        unsavedApp.setDateTime(APPOINTMENT.START_DT, startDt);
        unsavedApp.setDateTime(APPOINTMENT.END_DT, endDt);

        // save new appointment
        final DataObject savedApp = eimInterface.saveAndReturnObject(unsavedApp);

        // print identifier of appointment
        System.out.println("ID of new appointment = " + savedApp.getGGUID());
    }
}
public void documentTest() throws DataLayerException, BusinessException {
// 1st use-case: upload and test document

// create new appointment object (unsaved)
final DataObject unsavedDoc = eimInterface.createObject(DOCUMENT.value());

// set data to new document
unsavedDoc.setString(DOCUMENT.KEYWORD, "Outcome Meeting");

// create request object to upload a document
final CheckInFileRequest checkInFileReq = new CheckInFileRequest();
checkInFileReq.setDataObjectToSave(unsavedDoc.toDataObjectTransferable());

// set type of binary content (file ending)
checkInFileReq.setFileType("txt");

// set binary file content
final byte[] content = "It's all fine. \nThis is the file content.\n".getBytes();
checkInFileReq.setDocumentContent(content);

// execute request
final CheckInFileResponse checkInFileResponse =
    (CheckInFileResponse) eimInterface.execute(checkInFileReq);

// get saved document record
final DataObject savedDocument =
    new DataObject(checkInFileResponse.getSavedDataObject());

// print identifier of appointment
final String idOfSavedDocument = savedDocument.getGGUID();
System.out.println("ID of new document = " + idOfSavedDocument);

// 2nd use case, download binary document data
final CheckOutFileRequest checkOutFileReq = new CheckOutFileRequest();
checkOutFileReq.setGGUID(idOfSavedDocument);

final CheckOutFileResponse checkOutFileResponse =
    (CheckOutFileResponse) eimInterface.execute(checkOutFileReq);
final byte[] binaryContentOfDoc = checkOutFileResponse.getDocumentContent();

// print text file
System.out.println("content of document:\n" + new String(binaryContentOfDoc));
}
CONSORTIUM

CAS Software AG, Germany
Project coordinator: Dr. Bernhard Koelmel

UNINOVA – Instituto de Desenvolvimento de Novas Tecnologias, Portugal
Technical coordinator: Prof. Luis M. Camarinha-Matos

Universiteit van Amsterdam, Netherlands

iPLON GmbH The Infranet Company, Germany

Steinbeis GmbH & Co., Germany

SKILL Estrategia S.L., Spain

Komix s.r.o., Czech Republic

Prolon Control Systems, Denmark
Member of the:

www.glonet-fines.eu