

eHealth Framework

eHF Technical White Paper

Contents

1 Preface	3
2 Introduction	4
2.1 Document Overview	4
3 eHealth Framework Overview	5
3.1 Metaphors	5
3.2 eHealth Framework Elements	5
3.2.1 Development Environment.....	6
3.2.2 Application Platform.....	6
4 Conceptual Architecture	8
4.1 Primary Objectives	8
4.2 Modularization	9
4.3 Componentization	11
4.4 Module and Component Meta Model	11
4.5 Application Meta Model	13
4.6 Architectural Layers	14
5 Development Environment	16
5.1 Build Environment	16
5.1.1 Build Process	16
5.1.2 Continuous Integration	16
5.2 Code Generation	17
5.3 Runtime Environment	17
5.3.1 Web Server	17
5.3.2 Servlet Container	17
5.3.3 Database	17
6 Application Platform	18
6.1 Core Modules	18
6.2 Security Modules	18
6.3 Infrastructure Modules	19
6.4 Application Modules	19
7 Appendix	20
7.1 List of Abbreviations	20
8 References	21

Imprint

InterComponentWare
Industriestraße 41
69190 Walldorf
Tel.: +49 (0) 6627 385 0
Fax.: +49 (0) 6627 385 199

© Copyright 2008 InterComponentWare AG. All rights reserved.

Document version: 1.0
Document Language: en (US)
Product Name: eHealth Framework
Product Version: 2.9 Preview
Last Change: 27.04.2009
Editorial Staff: BAS Technology

1 Preface

The world-wide demand for health care has been and will be increasing steadily. So is the demand for electronic health care solutions. Medical data is recorded, processed, and interchanged electronically. Software-aided solutions complement and supersede traditional paperwork approaches. Stakeholders' ever-increasing demands and expectations lead to more sophisticated and complex software systems, while at the same time consumers and patients raise their voices and call for privacy and data protection.

eHF tries to position itself exactly in this context as the leading platform for developing electronic health care solutions. It empowers software developers to build state-of-the-art eHealth applications by providing reusable software components, development tools, and architectural guidelines and conventions, covering the full software development and product life cycle.

All logos are a registered trademark of ICW AG. The product names mentioned in this documentation are either the trademarks or registered trademarks of the respective owners and are stated for identification purposes only.

This documentation and the software components are protected by copyright © 2009 InterComponentWare AG.

Author: Karsten Klein

All rights reserved.

2 Introduction

The eHealth Framework (eHF) is a powerful platform for the development of health care solutions. InterComponentWare AG (ICW) has incorporated its extensive experience in developing and deploying solutions into the eHealth Framework, which represents the foundation of the Java Platform Enterprise Edition (Java EE) development at ICW. The eHealth Framework has been leveraged to allow development by external partners – enabling adopters a straightforward integration into ICW solutions.

The eHealth Framework consists of reusable software components, development tools, as well as architectural guidelines and conventions defining a full software development and product life cycle. From the perspective of a partner, the framework provides services and infrastructure capabilities for integrating applications within an eHF-based solution.

2.1 Document Overview

The structure of the eHealth Framework (eHF) Technical White Paper is adapted from the eHF Reference Documentation. The white paper has four major chapters, beginning with chapter three "eHealth Framework Overview", illustrating which components the development environment and the application platform consist of.

Chapter four gives insights into the conceptual architecture on an abstract level, not including technology and implementation details.

The development environment and infrastructure are described in the fifth chapter.

Finally chapter six provides an introduction to the application platform and its logical separation in four module categories core, security, infrastructure and the actual application modules.

3 eHealth Framework Overview

The eHealth Framework (eHF) is a holistic approach to developing electronic health care solutions. It provides tools, interfaces, software modules, and documentation that assist you throughout the entire software development and product life cycle.

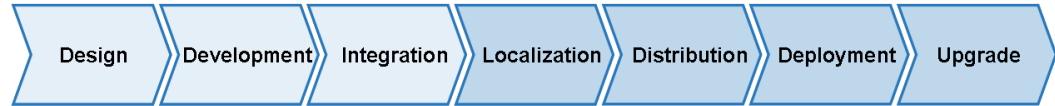


Figure 1: eHF supports the entire Software Development and Product Life Cycle

3.1 Metaphors

In order to introduce to the ICW eHealth Framework and to phrase its purpose and goals the following metaphors are utilized. They provide different perspectives on the eHF and mention various keywords motivating the central concepts of the framework.

The ICW eHealth Framework is the response to the latest industry challenges and trends.

In respect to this metaphor the eHF has to deal with industry standards and well accepted technologies. Furthermore methodologies derived or implied by the aforementioned trends have to be manifested.

The ICW eHealth Framework is an optimized infrastructure for the development, hosting and maintenance of applications and solutions in the health care sector.

In order to provide an optimized infrastructure the eHF has taken a holistic approach to developing software in health care. It supports the complete software life cycle from conception and design to deployment and maintenance. Accompanying the whole development and deploy cycle, eHF is able to identify opportunities for improvement and can eventually optimize the complete software supply chain.

The ICW eHealth Framework offers a kick-start advantage for the development of health care solutions in an environment with high demands.

In the context of this metaphor the various non-functional requirements can be introduced. Up front security and data protection are of central importance in the eHealth domain and - in combination with country specific legal restrictions or specifics - indicate a high demand on the software and its extensibility. Furthermore - apart from the software life cycle perspective that was already mentioned above - this metaphor leads directly into information life cycle concerns. In health care the lifetime of data may easily reach to the lifetime of a person and even beyond. The eHF has to anticipate this fact into the holistic approach and enable and optimize handling this concern.

3.2 eHealth Framework Elements

The eHealth Framework (eHF) consists of two main elements, as depicted in [Figure 2](#). On the one hand, there is the **development environment** which provides tools to design, develop, build, deploy, and run eHF applications. On the other hand, the **solution platform** offers you a framework and reusable software components that your products can build upon.

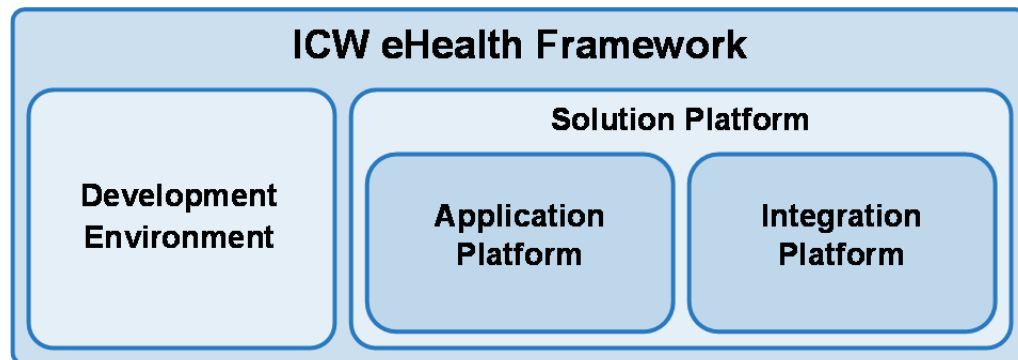


Figure 2: eHealth Framework Solution Development Platform

3.2.1 Development Environment

The eHF Development Environment is your all-in-one productivity platform for developing applications with the eHealth framework.

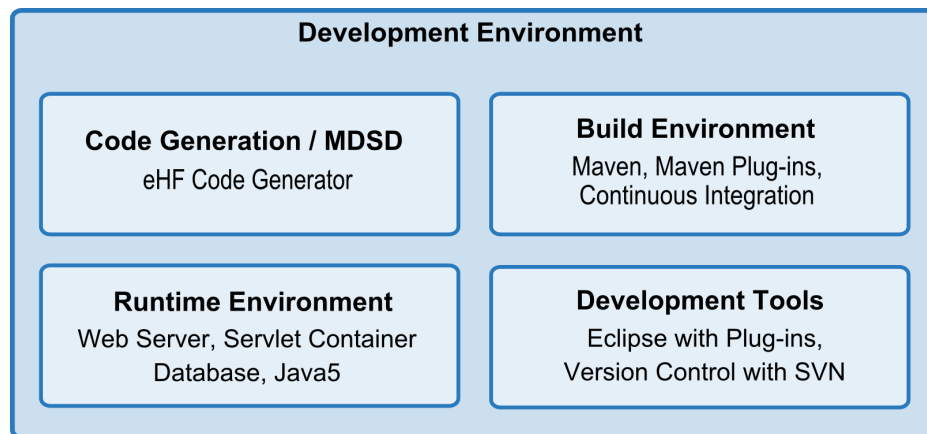


Figure 3: eHealth Framework Development Environment

It's most prominent part is the integrated *modeling and development environment* based on the Eclipse platform. The eHF *Code Generator*, driven by openArchitectureWare technology, boosts developer productivity by eliminating repetitive coding tasks. Build, assembly, and deployment are taken care of by the sophisticated *build infrastructure* based on Apache Maven. And finally, the eHF development platform bundles a comprehensive *runtime environment* that is required to run eHF applications.

3.2.2 Application Platform

The eHF solution platform is at the heart of all applications that you develop with the eHealth Framework. It is divided into the application platform and the integration platform.

The *application platform* consists of ready-to-use, modular building blocks. These range from fundamental core, infrastructure, and security modules to high-level, data-centric health care-specific application modules. They provide application programming interface

(API) functionality, which can be used by health care applications or custom add-on modules. Modules can expose web service interfaces that can participate in an orchestrated service-oriented architecture (SOA) infrastructure.

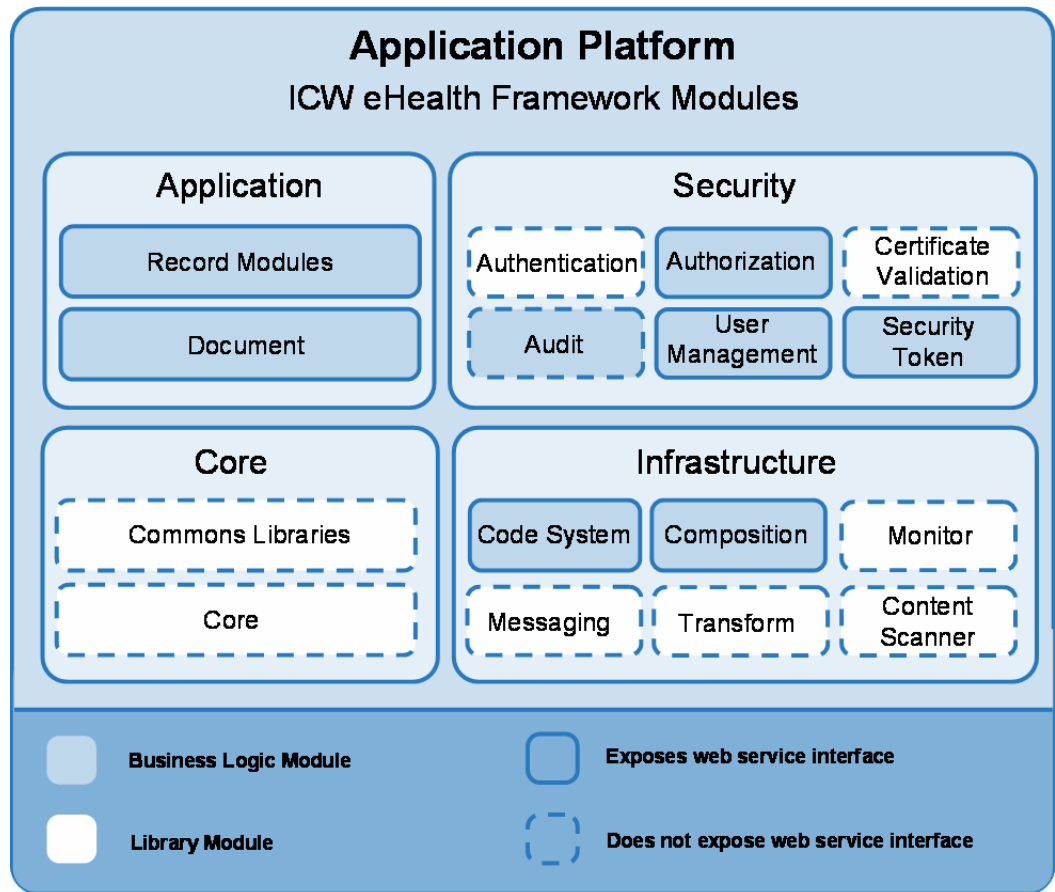


Figure 4: eHealth Framework Application Platform

The *integration platform* puts eHF applications in a broader, enterprise-wide perspective. To that end, it provides an enterprise service bus (ESB) as the technical infrastructure for enterprise-scale integration scenarios. The eHF integration platform is ongoing work and publishes its own reference documentation at this point in time. Please inquire for more information on the integration platform.

4 Conceptual Architecture

In this chapter the architectural principles and concepts are detailed. The main purpose is to describe the architecture on an abstract level, not to include technology and implementation details.

4.1 Primary Objectives

The eHF architecture, design and concepts are driven by the following objectives. These objectives are largely derived from non-functional requirements applications in health care have to meet. They are further ordered in terms of priority to emphasize the explicit needs in the demanding health care environment.

Application Security

Disposition: / Status: / Date-Time: / Author: MOE

Do you really want to restrict the security mechanisms to "health records"? I would propose to change it to "health care applications".

Enterprise-class authentication and authorization mechanisms ensure a very high level of security for sensitive medical and personal data in health records. The access of users to individual data objects is restricted according to the granted access rights.

Data Security

In addition to security on the application level the protection of data on the transport and database level is of central importance. The eHF provides sophisticated concepts on how to protect data from unauthorized access and theft.

Extensibility and Integration

Disposition: / Status: / Date-Time: / Author: MOE

I would propose to delete the second sentence. "requires to support" should be replaced with "supports".

The eHF architecture supports fast and efficient development of new modules, services, and applications. This development can be done by ICW or partners. The architecture requires to support straightforward integration of eHF services into heterogeneous existing health care environments and higher-level business processes.

Modularity

Modularity supports the distributed and loosely-coupled development of application modules by ICW and partners. The eHF architecture encourages and promotes the reuse of modules, components and services. This leads to a significant increase in speed and flexibility during the development process. Complexity, overall costs, and time-to-market for new or adapted functionalities are reduced.

Layered Architecture

To separate vertical concerns, different abstraction layers are required. These layers must be well-defined and access to other layers of the system must be narrowed and controlled, isolating technical details of adjacent layers. Therefore, bypassing layers is generally not allowed. A layered architecture enables a higher flexibility concerning improvements or replacements of technologies used in an individual layer without affecting other layers.

Separation of Concerns

In addition to architectural layers that isolate concerns vertically, general cross-cutting concerns are addressed using aspect-oriented programming (AOP) methods. In other

words, business logic should be strictly separated from concerns, such as security logic or transaction management. Architectures supporting AOP allow for better reusability of application components and easier maintainability.

Scalability and High Availability

The architecture scales from small user numbers to several million users. A system can be deployed to support high availability. This means that the system is not dependent on any single point of failure and that maintenance of the application will not affect availability.

Testability

The architecture supports fully automated functional testing on different levels (unit, module, service, application, solution). Furthermore, the architecture enables automated regression-, compatibility and upgrade tests.

It is a fundamental prerequisite for testing that the APIs in an application are well-defined, documented and exposed very consciously.

Portability

The architecture of the eHF allows for the smooth porting between different Java EE containers. Portability is further promoted by an orientation towards Plain Old Java Objects (POJO) programming models.

Standard Technology

The eHF is based on open source frameworks which are widely accepted and can therefore benefit from the experience and skills of a constantly growing community. This attracts partners who are familiar with these technologies and flattens what would otherwise be a steep learning-curve.

Serviceability and Maintainability

Software in a production environment has to satisfy strong service level requirements. The eHF defines standard procedures and tools for operating, maintaining and managing an eHF-based application.

4.2 Modularization

Modularity, extensibility, a layered architecture, separation of concerns, testability and portability are strong objectives, which significantly influence an architecture. There are various levels on which to approach these objectives and the concept of modularization is the most coarse-grained level to start with. A module isolates a functional concern and focuses on a specific domain. A module is connected to others on a distinct architectural layer and can be tested by consuming the functionality exposed by the module. A modularized application is by definition extensible as you can simply add an additional module to it.

Disposition: / Status: / Date-Time: / Author: MOE

I would propose to delete "which is it's opposite". I am not sure what you want to say with this and it seems to be grammatically wrong.

Apart from the fact that modularization is the concept of partitioning your system into functional units it has further advantages and benefits over a monolithic approach, which is it's opposite.

Clarity of Overall System Design

When organizing all your functionality into modules your overall system design is much cleaner and more structured than in other approaches. The modules and their functional responsibility is easy to grasp and to communicate. Using a combination or sequence of modules you can easily illustrate functional use cases as they are processed in the system.

Interface-based Access

Disposition: / Status: / Date-Time: / Author: AKA

Wouldn't a title like 'Information Hiding' or 'Implementation Hiding' be a better title? Interface-based is just the way we implement it.

Modularization - if manifested consistently and forcefully - only allows access to functionality using defined interfaces. A consumer of a module's functionality cannot access internal parts of the module. He is forced to make use of the selected and exposed functionality through interfaces the module defines at its boundaries.

Disposition: / Status: / Date-Time: / Author: MOE

The third sentence does not work at the end. Do you mean: "tools ... to assure/enforce that these boundaries are not violated"?

For this to work a clear definition of the module boundaries and a technical enforcement is required. Otherwise violations are not prohibited and will be considered available. The eHF architecture therefore provides tools to define the boundaries of a module and to enforce this boundary is not violated.

Isolation for Evolution

Disposition: / Status: / Date-Time: / Author: MOE

Last sentence: I would propose to replace "destroying" with "counteracting".

By consciously defining the boundaries of a module the internal implementation details can be hidden. Hiding in itself cannot be regarded as an advantage here, but the fact that you can constantly evolve the internals of your module - without having to worry about destroying an undefined contract with a consumer - keeps you flexible.

Disposition: / Status: / Date-Time: / Author: MOE

The following paragraph repeats what has been described before: evolution of internals without counteracting the contract. There isn't any new information...

In this context, a principle of Domain Driven Design may also help to understand the benefits. In Domain Driven Design it is regarded as essential that you constantly evolve your understanding of the domain and simultaneously your code. Modularization and the concept of a defined public API help you to understand and eventually not to break your contract, while still being able to change the internals of your module.

Disposition: / Status: / Date-Time: / Author: AKA

Name the DDD principle: ubiquitous language

Isolation for Integrity

In order to implement a certain kind of functionality usually several components act in concert with each other and a request has to follow a certain path through these components in order to produce a consistent and valid result. Bypassing parts of the execution (for example, accessing it at a deeper level than anticipated) may endanger the integrity of the result and ultimately the system.

Modularization and enforcement of a public API prevent such a violation. The functional responsibility of the module cannot be compromised.

Public API Rules

Some rules can be extracted and derived with respect to the public API:

- It is prohibited to violate the public API of other modules.
- It is prohibited to bypass the public API of other modules.
- Direct access to the database schema of another module is prohibited.
- Any assumptions on a technical level that are not part of the public API of the module are not allowed.

Such rules and their consequences may appear artificial in some cases. For example, the limitation to access all data via the API, as mentioned above, does not seem to be appropriate. However, in the health care domain several access control and data protection

requirements have to be considered. The eHF is positioned in this domain and aims to enable strict authorization and protection rules. This results in the above rules, as data is only accessible via the public API of the module that implements the domain specific protection mechanisms.

Disposition: / Status: / Date-Time: / Author: AKA
Metaphor: no back-door access, everybody is forced to use main entrance.

Despite what seems to be a hard limitation, such a paradigm offers various advantages at the level of data protection and for software life cycle management.

As part of the eHF Development Environment the definition and enforcement of the public API of modules is supported.

Disposition: / Status: / Date-Time: / Author: AKA
Missing: Modularization allows for software reuse, organizational aspects (e.g. better scalability within development teams)









4.3 Componentization

While modularization is a coarse-grained approach to partitioning your system into functional units, componentization is another level of separating functional concerns.

In the eHF architecture components are defined as elements a module is composed of. Components normally belong to a distinct layer of a layered architecture.

Please note that the term component is highly overloaded in software development. The definition used in the scope of the eHF refers to functional units of a fine granularity. In many contexts the term can mean very coarse-grained functional units (which we would refer to as subsystems) or even bits and pieces of hardware.

The following table is a classification of components as defined in the context of the eHF.

	Structural Component: All kind of data structures are considered structural components.
	Behavior Component: Abstract base classes which define generic behavior and can be implemented from other modules.
	Configuration Component: Configuration files or configuration fragments of any kind.
	Functional Component: Self-contained units (utilities) which may be re-used from other modules if exposed.
	Persistence Component: DAO implementations or other persistence layer elements.
	Contract Component - Provider: Defines a contract.
	Contract Component - Consumer: Defines a required contract. This can be also understood as extension point or service provider interface (SPI) consumer.
	Integration Component: Component providing integration portions. Integration components can be combined to address an integration scenario.

4.4 Module and Component Meta Model

The eHF architecture provides functional separation using a decomposition in modules and components. In order to illustrate the relationship between modules and components the following meta model was derived. The meta model and the pictograms that are

used provide a powerful tool to describe the responsibilities, boundaries, relationships and interactions between modules.

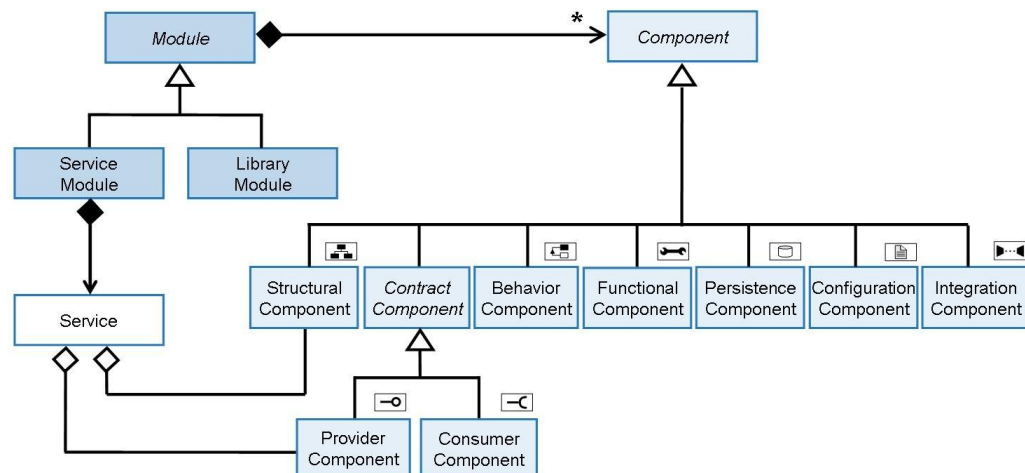


Figure 5: eHF Module and Component Meta Model

Figure 5 shows the eHF module meta model. According to the figure a module consists of several components. Modules are either library modules or service modules.

Library Modules

A library module provides an API or a set of abstract base classes with well-defined extension points that provide a common implementation basis for other modules. A library module can define service interfaces specified in the context of an API which can be implemented. It also offers APIs encapsulating access to basic library functions.

Library modules do not require a database and do not expose an external (web) API. Inside the eHF, library modules can be found in the core, security, and infrastructure domains. For example, library modules are responsible for common tasks such as authentication, messaging, or transformation.

Service Modules

Disposition: / Status: / Date-Time: / Author: AKA
 Term 'functional unit' confuses here. It is used as well when describing components.

Service modules are self-contained and isolated functional units. They are more complex components which encapsulate a specific localized domain and may expose their functionality externally as a web service or internally to other modules or any other software. A service consists of a provider component and structural components (parameter and result types).

Service modules can use functionality from both library modules and other service modules. They can implement business logic, which requires services of other service modules. Thus a complex business process provided as a single service can actually be a collaboration of several services from different service modules.

Module Interdependencies

Figure 6 shows the two module types and some possible interdependencies. Modules can extend other modules (behavior components), can use other ones (structural or functional components), can implement or consume them (consumer component) or can provide functionality to other ones (provider component). However, there is a constraint that service modules cannot be extended.

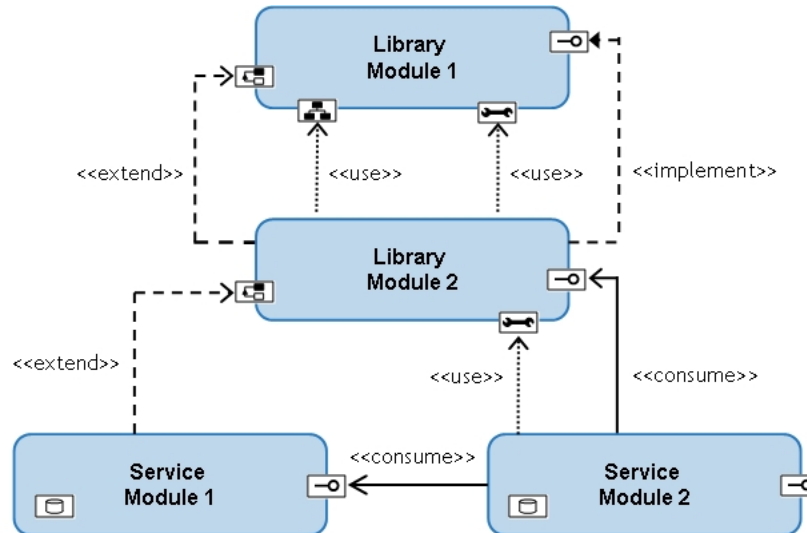


Figure 6: Module Types and Dependencies

Modules within the eHF either interact synchronously via procedure calls (request-response) or asynchronously via one-way messages. Message-based interactions are primarily used to communicate application events. Messages are sent via messaging services provided by the eHF Application Platform.

4.5 Application Meta Model

The following image shows the meta model for an application. Basically an application consists of a set of service modules, library modules and third-party modules that are composed using additional configuration components. The configuration components specify how the modules interact with each other. Furthermore, application specific behavior of the modules can be defined on this level.

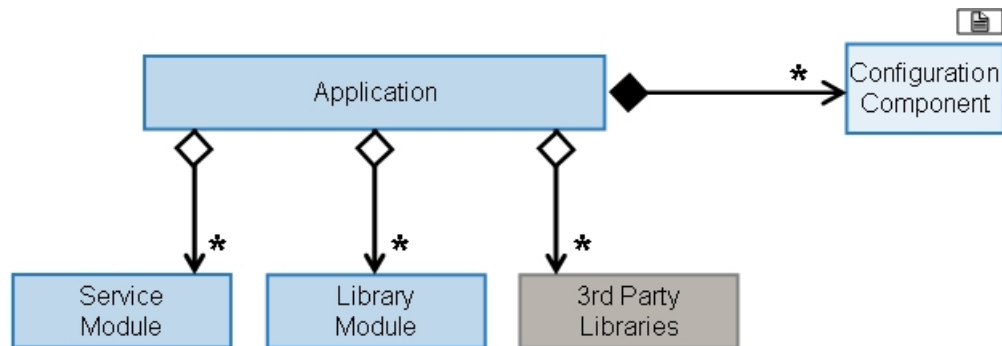


Figure 7: Meta Model of an eHF-based Application

The eHF Development Environment provides tools to define such an application. In this context we call a project to define an application assembly. The result of an assembly build or release is a distributable artifact that contains all the elements of the application.

The following figure shows a schematic instance of an application. The application exposes the web service provided by the modules. The web service of eHF Service Module n is specific to the application as it implements the desired service that the application requires.

eHF Service Module 2 consumes the service of a third-party module or service. The figure also indicates that eHF Service Module n does not have any persistence. This normally means that it is only a facade to existing modules with the only focus being to implement

an application-specific use case. The logic that is required is represented by the wrench pictogram.

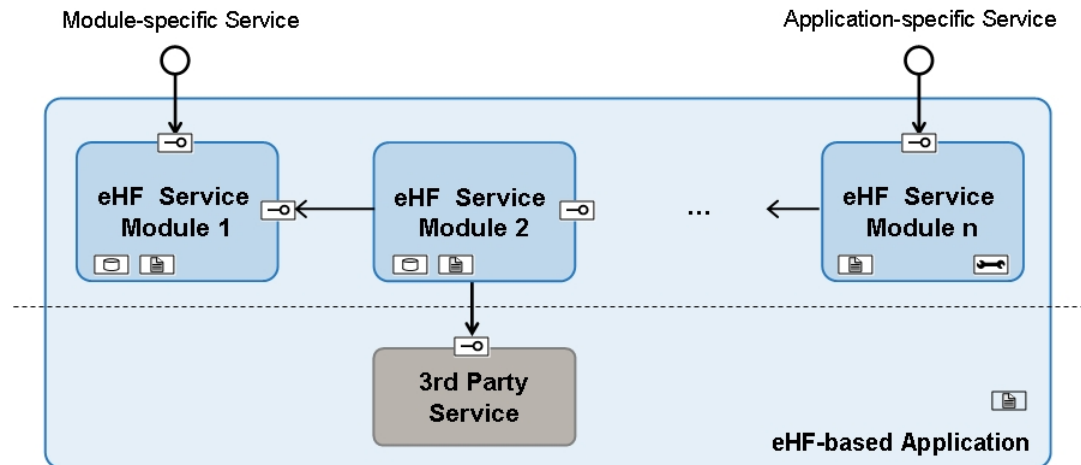


Figure 8: Example of an eHF-based Application Using the Component Pictograms

4.6 Architectural Layers

Next to modularization, *layering* constitutes the second cornerstone of the eHealth Framework's (eHF) architecture. While modularization focuses on functional, vertical (de-)composition, layering enforces a responsibility-driven, horizontal structuring of the software system. A module typically stretches across many, but not necessarily all layers.

Layering is one of the most common techniques used for structuring complex software systems. Each individual aspect of the software system is conceptually assigned to one of these layers. The dependencies between the different components are organized in such a way that only parts on higher layers may use parts on the same or lower layers. These layers have well-defined interfaces and controllable access to other layers of the system, thereby isolating technical details of adjacent layers. This layered architecture also allows for the possibility to upgrade or replace one or more tiers independently as requirements or technology change.

Conceptually the eHealth Framework's service stack consists of three layers; persistence, service and service adapter. In the persistence layer data is stored and retrieved. It keeps data neutral and independent from the business logic found in the service layer, which is exposed to clients by the service adapter layer.

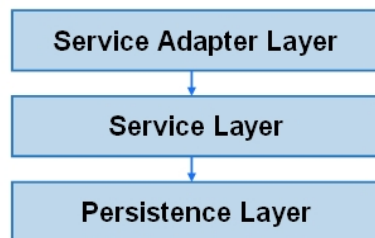


Figure 9: Module Service Stack

Persistence Layer

For the persistence layer the eHealth Framework uses the Data Access Object (DAO) Design Pattern to build a Data Access Layer (DAL). The service layer uses this DAL to access and manipulate stored data objects without having to deal with the complexities of database access. The DAOs define the interface between the service layer and

the concrete data sources. They hide persistence technology details and decouple the business layer from changes in the persistence layer.

Service Layer

The service layer is an architectural layer that separates the business logic from other components like the persistence layer or presentation layer. It aggregates the business logic in a central place, which promotes consistent behavior across different applications. In addition it hides complex logic from application clients ensuring an easier, faster and less error-prone development process.

The service layer accepts requests from the service adapter layer and interacts with the persistence layer in order to map the business objects to the database.

Service Adapter Layer

The service adapter layer provides well-defined interfaces for integration. It decouples the service layer from the presentation layer or from other applications that are using the eHF services.

5 Development Environment

The eHealth Framework provides both a runtime environment for health care solutions and an application development platform that can be used for the development of new as well as existing modules. The eHealth Framework supports and accompanies the full software development life cycle to produce lean and high quality health care applications.

Developers can turn to the provided development support and use it to enhance their productivity. The eHealth Framework supports agile and iterative development methodologies. When this approach is applied consistently, it will result in the creation of robust and sophisticated health care applications. The modules that are provided by the eHealth Framework have been developed using the eHealth Framework development environment.

5.1 Build Environment

5.1.1 Build Process

Module development requires a significant amount of cyclic routine tasks, which can easily be automated. The eHealth Framework supports such automated tasks in its build environment, which is based on the software build tool Maven.

Maven provides many plug-ins to deal with common tasks. The eHF supplements the Maven build environment with its own Maven plug-ins. The following list introduces the most relevant tasks supported by the eHF:

1. Generation Process
 - Code Generator performs model-to-code transformations. See section 5.2 for more information on the generator.
 - Hibernate tool for generating schema creation scripts
2. Data Handling
 - Database lifecycle, e.g. starting and stopping the database
 - Database bootstrap and import for initializing the database with bootstrap and test data.
3. Quality Assurance Reports and Documentation
 - Source code analysis for control of code consistency (for example Check-Style).
 - Unit tests to perform module unit tests (for example JUnit).
 - Regression Tests to perform functionality regression tests (for example WebServices)
 - Test Coverage to analyze test coverage (for example Cobertura) metrics.
 - JavaDoc to obtain complete module API documentation.
4. Product Build
 - Assembly to integrate several modules into a distributable artifact
5. Product Deployment
 - Deployment to deploy an assembly artifact on a target system. A deployable (configured for the target system) artifact is created from a distributable artifact.
 - Updates and Patches functionality to install updates and patches.
 - Upgrade sophisticated support to upgrade an existing product, considering schema and data changes in the database.

5.1.2 Continuous Integration

The eHealth Framework provides tools and guidelines which enable partners to easily implement a fully automated continuous build process. By utilizing the continuous integration toolkit CruiseControl, the version control system (CVS, Subversion) is constantly monitored for changes. Any change in the version control system executes the build process (ANT, Maven) including all tests (JUnit). Consequently, the continuous integration process runs several times a day, thereby detecting and reporting errors as early as possible. An inter-module integration test is performed in order to further ensure stability.

5.2 Code Generation

Model-Driven Software Development (MDS) is the backbone of the eHealth Framework's development lifecycle support. The Code Generator component is able to generate a significant amount of configuration, code and other module-specific artifacts. The approach is both domain and architecture driven. This means that the main generator input is a UML domain model. From this model a full-fledged and ready-to-use skeleton of the module can be generated, which covers all required architectural layers and reusable artifacts. The generator offers well-defined extension points marked with task comments in the generated sources.

The generator automates the creation of the full persistence data model, the appropriate service layers and the integration layers. Custom service methods can be added as required. The generator approach abstracts from the technical details of the architecture, emphasizing the domain model, the accompanying business logic and services. The eHF Generator is based on the openArchitectureWare open source generator framework. It is highly configurable and can be customized to meet specific needs.

5.3 Runtime Environment

A mature runtime environment is necessary in order to enforce an efficient development lifecycle for health care applications. A reference deployment infrastructure based on open source components is used for the eHealth Framework.

5.3.1 Web Server

The eHealth Framework reference deployment utilizes the Apache HTTP server. For development purposes, this web server acts as Secure Socket Layer (SSL) endpoint for secure communication. Furthermore, it provides an Apache JServ Protocol (AJP) Connector as a bridge to the servlet container. Thus, it represents a single point of access for development.

5.3.2 Servlet Container

The Tomcat Server is used as the servlet container during the development process. The Tomcat instance provided with the eHealth is already shipped with a special security configuration for eHealth Framework modules. All assembled eHealth Framework modules can be deployed and tested within this servlet container.

5.3.3 Database

The eHealth Framework development environment uses a lightweight Hypersonic SQL (HSQLDB) database engine that provides persistence functionality.

6 Application Platform

The eHF application platform consists of business service modules and library modules that are built on top of industry-standard application frameworks and can be deployed on any J2EE 1.4-compliant application server.

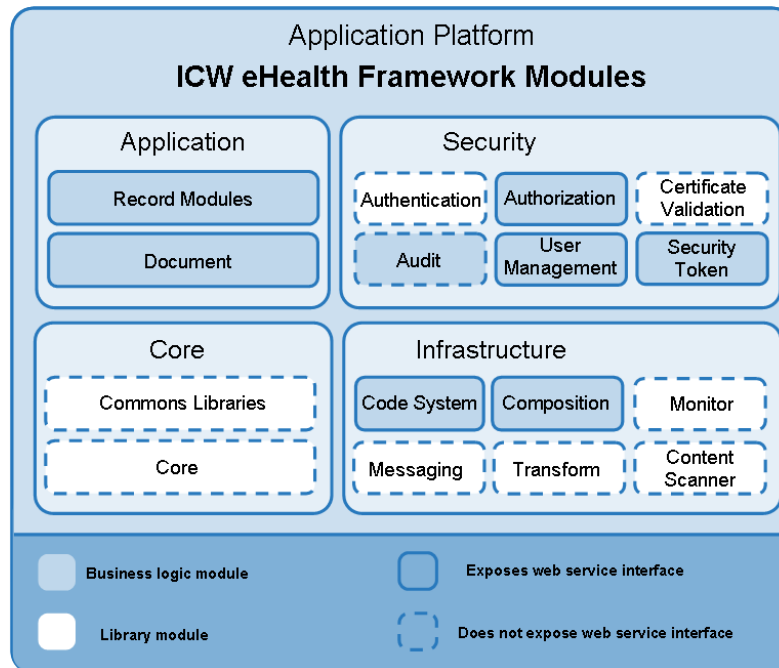


Figure 10: Application Platform Modules

The eHF modules are organized into four categories. [Figure 10](#) shows the categories Core, Infrastructure, Security and Application.

6.1 Core Modules

The Core modules establish the platform for other modules and provide basic functionalities. The Commons library module provides infrastructural features, which are used as a basis by individual business service modules. The Core library module hosts a general domain model, which is used throughout the framework.

6.2 Security Modules

Sophisticated services to address the sensitivity of medical information (privacy, integrity and availability) are provided by the modules of the Security category.

The **Authentication** library module verifies the identity of users or external systems. Only authenticated requests are allowed to interact with eHF services.

The **Authorization** module manages access rights (permissions) and controls access to protected resources. Its services implement hierarchical role-based access control (hierarchical RBAC) mechanisms, including instance-based access control features and black list / white list functionalities.

The **Audit** module implements means to store audit information of security relevant actions. The auditing of events is critical for satisfying specific regional data security requirements. The User Management module can be used by a health care application to manage the life cycle of its userspecific information. The administration of user roles is supported as well.

Further security-related modules explained in this chapter are

- **certificate validation,**
- **content scanner,**
- and **secure token services.**

6.3 Infrastructure Modules

The category encompasses modules with different but important responsibilities concerning for example inter-module communication realized by the Composition Module.

The Code System Module implements the technical concept of controlled vocabularies, dictionaries and classification systems. Therefore it provides an API to query and resolve code systems. It also supports the exchange of standardized information and localization.

The Code System Repository contains codes information such as administrative data (e.g. country, language, marital status), medical terms (diagnosis type, encounter etc.) or system or error messages.

6.4 Application Modules

The Application modules allow the maintenance of medical information. The module eHF Document enables storage and retrieval of documents of arbitrary content. It supports a specialized document versioning mechanism. The module eHF Record provides, on the one hand, services for basic medical information such as diagnoses, medications, and observations, and on the other, administrative data that are related to patients and emergency contacts.

An other important module in the application context is the Document Module. In the health sector more and more medical information is stored in electronic documents. Examples of such documents are discharge letters, medical histories, ambulatory visit reports or X-ray photographs. In order to manage such documents, systems have to support tasks such as storing, retrieving, organizing or finding documents. The application module eHF Document is responsible for document management within the eHealth Framework and should be part of any health care application built on top of eHF.

7 Appendix

7.1 List of Abbreviations

Abbreviation	Definition
ANT	Another Neat Tool
AJP	Apache JServ Protocol
AOP	Aspect Oriented Programming
API	Application Programming Interface
CVS	Concurrent Version System
DAL	Data Access Layer
DAO	Data Access Object
DTO	Data Transfer Object
eHF	eHealth Framework
ESB	Enterprise Service Bus
HSQLDB	Hypersonic SQL Data Base
ICW	InterComponentWare
MDSD	Model Driven Software Development
POJO	Plain Old Java Objects
RBAC	Role Based Access Control
SOA	Service Oriented Architecture
SPI	Service Provider Interface
SQL	Structured Query Language
SSL	Secure Socket Layer
SVN	Subversion

8 References

ICW Developer Network

<http://idn.icw-global.com/> ↗

ICW eHealth Framework Reference Documentation

BAS Technology and eHF Committers, InterComponentWare AG (2008-2009)

<http://idn.icw-global.com/downloads.html> ↗