and XML, which represents a technique that combines JavaScript, the Document Object Model (DOM) and the XMLHttpRequest approach in order to make Web applications more dynamic and responding. Ajax enables to perform asynchronous partial updates of web page components using Ajax engines. If the user initiates an action on a page, he does not communicate with a server through a HTTP request. He interacts with the Ajax engine per JavaScript instead, which processes all operations, which resulted from the interaction of user with the web application. If an operation does not need a connection to the server, the Ajax engine handles it by itself. For operations that require a client-server communication, the Ajax engine directly connects to the server. Therefor, it sends an asynchronous request to the server, so the user interface does not have to be reloaded as a whole [13].

JSF 2.0 provides Ajax functionality in form of the build-in resource library, which can be applied to UI components using the *<f:ajax>* tag to enable Ajax requests. As a result, only the UI components including the *<f:ajax>* tag will be submitted, validated and rendered without performing all lifecycle phases for the whole view [20].

A further possibility to apply the Ajax functionality to a particular UI component is to use the *update* attribute. The value of the *update* attribute contains the ID of the UI component, which has to be updated.

2.3. Gargoyle-Codegenerator

In the implementation part of this bachelor thesis the Gargoyle Codegenerator was used to automatically generate basic building blocks of the business logic (EJB source code). The process also included the generation of JSF visualization source code that consists of managed beans and XHTML-pages. The target architecture of the Gargoyle Codegenerator is based on an architecture developed at the LuFGi3 at RWTH Aachen, which is shown in figure 2.3. The architecture is divided into three layers: the client layer, the business layer and the data layer. The client layer represents the user interface, implemented by JavaServer Faces. Methods of the managed bean classes access an application facade in the business layer, which implements the EJB technology. The application facade calls the methods of the CRUD (Create, Retrieve, Update, Delete) classes and includes the corresponding methods for creating, retrieving, updating and deleting of application objects. The application facade and the CRUD Controller implement the business functionality of the whole application [31].

In order to access the persistence layer the methods of the CRUD classes call DAOs. DAO is an abbreviation for data abstraction objects, which represent abstract objects of the data layer. The domain model describes the business objects of the application, which is realized through the DAO methods and the

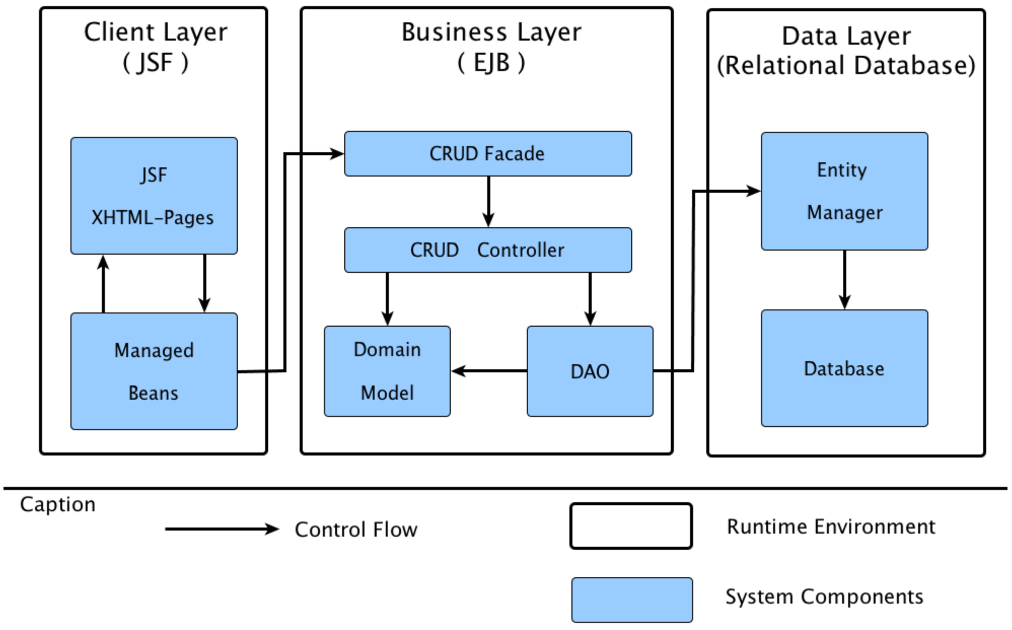

Figure 2.3.: The Target Architecture of the Gargoyle Codegenerator.

entity manager.

The architecture of the Gargoyle Codegenerator itself is designed as illustrated in figure 2.4. Its implementation structure allows the user to control each generation step by using several transformation models. As shown in figure 2.4, there are six model transformation steps, which consecutively generate components of the target application architecture. The transformation process includes two types of transformations, the Model-to-Model transformation and the Model-to-Text transformation. In figure 2.4 the Model-to-Model transformation is represented by the straight arrows and the Model-to-Text transformation is indicated by the dashed arrows [31], [24].

The code generation needs an UML model as input data. The UML model will be imported, which is denoted by the red dashed arrow. From the UML model a generator model is created, which defines entities of the domain model and their relationships. In the generator model particular information for the generator are added.

The entity model is created from the generator model by using the Model-to-Model transformation. The entity model is a preparation step for the generation of the domain model classes and the JPA. Therefor the entity model extends the classes of the generation model by ID attributes and database named queries needed by JPA annotations. The creation of the domain model source code from the entity model is performed by using a Model-to-Text transformation.

The entity model is also a start model for the EJB - Generator model, that

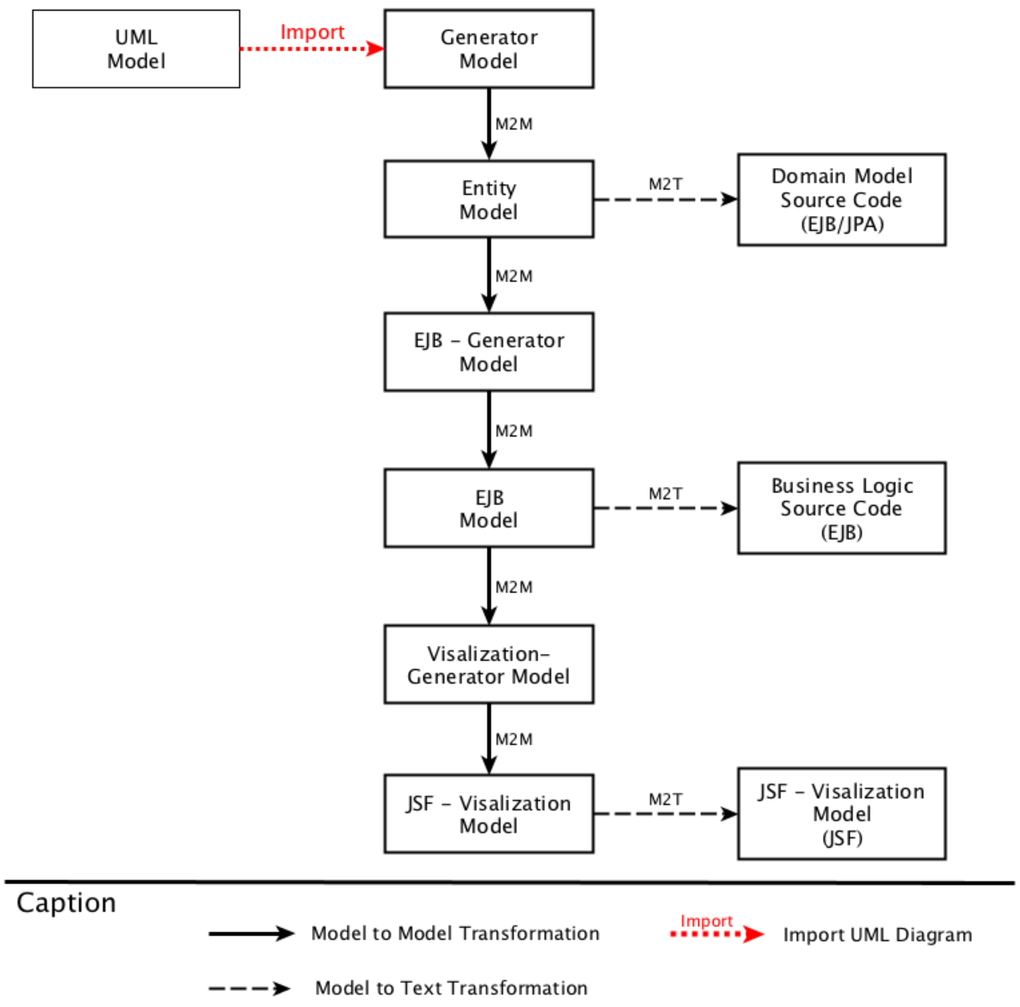
UML Model
Import
Generator Model
M2M
Entity Model
M2T
Domain Model Source Code (EJB/JPA)
M2M
EJB – Generator Model
M2M
EJB Model
M2T
Business Logic Source Code (EJB)
M2M
Visalization-Generator Model
M2M
JSF – Visalization Model
M2T
JSF – Visalization Model (JSF)
Caption
Model to Model Transformation
Import
Import UML Diagram
Model to Text Transformation

Figure 2.4.: The Gargoyle Codegenerator Architecture.

defines the CRUD components and methods. From the EJB - Generator model the EJB model is created, which includes the essential elements for the business logic implementation: DAO, the CRUD Controller and the CRUD Facade. The CRUD Controller and Facade implement the methods which specify the functionality of the application, while the DAO classes provide methods to access the database. When the methods of the EJB model are adjusted and edited, the source code for the business logic can be generated.

The generation of the JSF code will be executed in two steps. As shown in figure 2.4, the first step is performed by using the visualization generator model. Here a connection to the Facade, the specification of methods to access the Facade and the page relationships are defined. The visualization generator model is the basis for the JSF visualization model obtained by another Model-to-Model transformation. In the JSF visualization model, the managed bean classes and the JSF template pages (Facelets) are described. In the last step the JSF source code for the managed beans and the Facelets is generated [9], [24].

2.4. Metrics

As defined in the title, the main task of this bachelor thesis deals with the extension of a dashboard functionality. A project dashboard can be defined as a tool for the visualization of different project metrics (see section 2.5). Therefor metrics are an important basis of this bachelor thesis.

The term 'metric' is intuitively related to the terms 'measure' or 'measurement.' Hence, the term 'metric' is often explained in literature by using the terms 'measure' or 'measurement' and vice versa [30]. However, measurement is defined in ISO/IEC 15939 as a *"set of operations having the object of determining a value of a measure."* [19] The IEEE defines the term 'metric' as *"a quantitative measure of the degree to which a system, component, or process possesses a given attribute."* [17] Hence, a metric is a mapping of a measured property to a scalar or vectorial scale [25], while a measurement is simply a value.

Metrics are an objective assessment of an artifact and determine the quantitative characteristics of an artifact. This means, that the artifact characteristics are mapped to a numeric value or a vector of numeric values. Different artifacts have a different degree of structuring. It is more difficult to represent poorly structured artifacts in a metric than the well-formalized artifacts [3].

Referring to project development, metrics are *"objectively measurable attributes"* [8] used to provide information about the project state. In the management area metrics are important *"decision-support tools"* [10] used to validate, control and report collected data. They help to find an effective and efficient direction of the development and organization process and make it more predictable.

Software Metrics

In software engineering the term 'metric' is extended by the term 'software metric' and uses to determine quality properties of software system and the software development process during its lifecycle. Software metrics help to estimate the required quality properties like maintainability, efficiency, flexibility, portability etc. as well as software development costs, project scheduling or risk register [8].

In [25] an 'ideal' metric is defined as a metric, which provides an accurate characterization of a measured software object, restricted to the measured property. The authors define seven main requirements for the software metrics: differentiability, comparability, reproducibility, availability, relevance, profitability and plausibility. They also present the following categorization of the software metrics, which are classified according to their application area.