Data Model Driven Implementation of Web Cooperation Systems with Tricia

Thomas Büchner, Florian Matthes, and Christian Neubert

Technische Universität München, Institute for Informatics,
Boltzmannstr. 3, 85748 Garching, Germany
{buechner,matthes,neubert}@in.tum.de
http://wwwmatthes.in.tum.de

Abstract. We present the data modeling concepts of Tricia, an open-source Java platform used to implement enterprise web information systems as well as social software solutions including wikis, blogs, file shares and social networks. Tricia follows a data model driven approach to system implementation where substantial parts of the application semantics are captured by domain-specific models (data model, access control model and interaction model). In this paper we give an overview of the Tricia architecture and development process and present the concepts of its data model: plugins, entities, properties, roles, mixins, validators and change listeners are motivated and described using UML class diagrams and concrete examples from Tricia projects. We highlight the benefits of this data modeling framework for application developers (expressiveness, modularity, reuse, separation of concerns) and show its impact on user-related services (content authoring, integrity checking, link management, queries and search, access control, tagging, versioning, schema evolution and multilingualism). This provides the basis for a comparison with other model based approaches to web information systems.

Key words: data modeling, web framework, web application, software engineering, software architecture, domain specific language

1 Motivation and introduction

Developers of enterprise web information systems and social software solutions are faced with a complex technology stack of programming languages (Java, PHP, Python, ...), persistence managers (Hibernate, JPA, JCR, ...), authorization and access control frameworks, template engines (Servlets, JSP, ...) and web form validation solutions. In order to support access via web APIs from third-party applications or stateful, rich, mobile clients (iPhone, Android, ...) even more technologies have to be employed.

These software development approaches suffer from the fact that small changes in the customer requirements (e.g., adding an attribute to a persistent entity, changing the cardinality of an association or changing the access policy for a certain user group) lead to numerous changes in all the layers of the server and

even on the client (e.g., JavaScript code for AJAX validation). These changes are very error-prone because of the mismatches between the type systems and data models involved (object-oriented, relational, tree-based). Based on our industrial experience of the last ten years implementing content management solutions, knowledge management solutions and community platforms we have developed during the last three years a data model driven approach which is supported by Tricia, an open source Java platform [13, 3].

The core of Tricia is an innovative modeling language tailored specifically to the needs of this problem domain. The main idea is to derive all necessary boilerplate implementation details from model-representations (data, access control, interaction) available at runtime as Java classes and objects. The application developer can thus focus on the pure application-specific business logic which is implemented in Java with language (typing, binding, scoping) and IDE support (auto-completion, refactoring, dependency checking, ...).

The purpose of this paper is to give an overview of the Tricia software architecture and development process (Section 2) and to present and motivate the concepts of its domain-specific data model (Section 4). In Section 3 we use the running example of a small Wiki application that allows end users to create wikis with wiki pages, comments, tags, etc. Throughout the text we highlight the benefits of this particular data model for application developers and for end users. This is the basis for a comparison with related work on model driven implementation of web applications in Section 6. We present exemplary views on the data model generated by model introspection (Section 5). The paper ends with concluding remarks and points to subsequent publications which will describe Tricia's access control model and interaction model that builds on the data model described in this paper.

2 Overview of the Tricia software architecture and development process

Figure 1 provides an architectural overview of a typical web application implemented on the Tricia platform using a notation similar to an UML deployment diagram. Such an application provides HTTP(S) access for its web clients (possibly including AJAX-style asynchronous interactions), a REST-ful web API to allow third parties to query and update the content managed by the application and a *Model Introspection* interface to allow third parties to discover and query the data model, access control model and interaction model implemented by the application. Our current implementation only supports a single server (up to fifteen page requests per second on stock hardware) but the architecture is designed for a scale-out to multiple servers using a cluster database.

A Tricia application requires a Java 1.6 runtime environment on Windows or Linux, a database server, and a Lucene full-text search engine. Currently Tricia supports MySQL, Oracle, and for testing purposes the in-memory database HSQLDB. There also exists a prototypical implementation which persists data using the NoSQL database MongoDB.

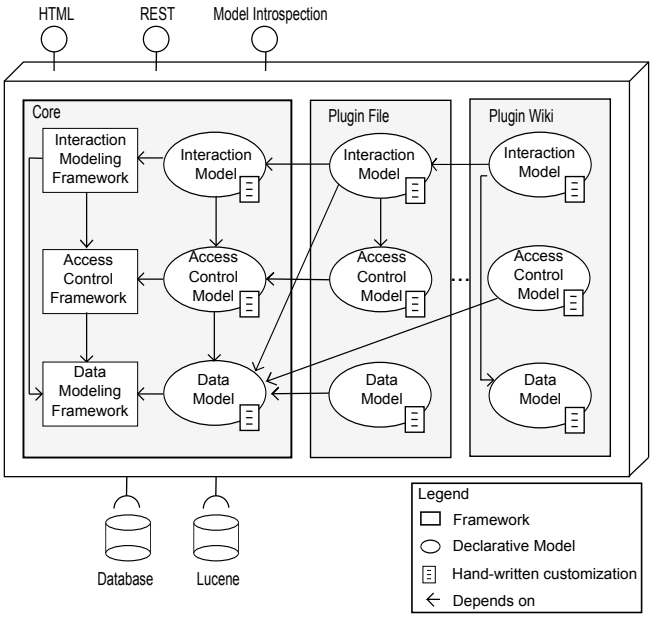

Fig. 1. Architectural overview of a typical web application implemented on the Tricia platform

A Tricia application consists of a *core* and one or more *plugins* that define the application in a modular fashion (see shaded areas in Figure 1). Each plugin specifies the plugins it depends on. Cyclic plugin dependencies are not allowed and are detected at construction time. For example, the plugin `Wiki` depends on the plugin `File`, since files may be attached to wiki pages and wiki management thus requires file management.

The core defines abstractions required by virtually all applications of the domain, for example user profiles, groups, memberships, login and registration procedures. The plugins `Wiki` and `File` both use such user profiles to identify the last editor of a wiki or a file. Other abstractions provided by the core are discussed in Section 4.4.

Each plugin and the core define a data model, an access control model and an interaction model. Each model defines a fragment of the data structures and behavior of the entire application. These models are expressed by graphical and textual notations (see Section 4) and are available at runtime for introspection (ovals in Figure 1). If necessary, they can be augmented by customizations written in Java (e.g., to express business logic). Models from a plugin P may reference models in P and in plugins imported by P (depicted by arrows in Figure 1). The following rules apply: Interaction models may reference other interaction models, access control models or data models. Access control models may reference other access control models or data models. Data models may refer-

ence other data models only. The core is provided as part of the Tricia platform and consists of three layered Java frameworks (c.f. layered architecture in [8]) for data modeling, access control and user interaction (views and controllers). Each framework provides abstractions and extension points, which have to be instantiated or customized in order to build a Tricia application. Frameworks are developed and maintained by the Tricia core developers as part of the core development process, customizations are developed by application developers as part of the application development process [12]. There are two different kinds of customizations. The majority of customizations can be done in a declarative, model driven way. This results in models to be created. For some aspects to be customized it is more convenient to specify them using the full expressive power of the base language, which is Java in our case. An example for this kind of customization is complex business logic. Figure 1 emphasizes the central role of the data modeling framework as the foundation for model driven web application development. Due to space limitations we focus in this paper on the concepts of the data modeling framework. We plan to describe the other frameworks and their meta models in subsequent papers. The following examples should suffice to highlight the use of the application-specific data model in all frameworks:

- For each entity type, the Tricia interaction framework can generate multilingual element-oriented CRUD views (create, read, update, delete) and set-oriented table controls. These views may include rich text attributes and media attachments (images and files).
- Associations between entities can be navigated in an element-oriented (via hyperlinks) or set-oriented (declarative queries) style. End users can interactively create full-text and structured queries for entities of a given type or any type (Google-like searches).
- Tricia can also expose these views and controllers as REST-ful web APIs to allow external systems to interact with Tricia applications.
- The Tricia access control framework allows application developers or end users to associate access control policies with entity types or even individual entities. These policies can restrict read, write and administration rights to user groups or to individual users (role based access control or discretionary access control). The policies are enforced automatically at the user interface and at the web API level.
- The Tricia data modeling framework automates the data migration steps necessary after (series of) typical incremental schema changes.

3 A small sample application

In the following, we will introduce the concepts of the Tricia data model step by step using a simple sample application, which allows registered users to manage a collection of wikis. Each of these wikis contains multiple wiki pages. One of the pages of a wiki can be specified as the wiki home page. Wikis and wiki pages are identified by a unique name and a readable, structured and persistent URL.

The content of a wiki page is a rich text (with markup, embedded hyperlinks and attached media files). This Tricia application consists of a plugin with a data model that defines the entities `Wiki` and `WikiPage`.

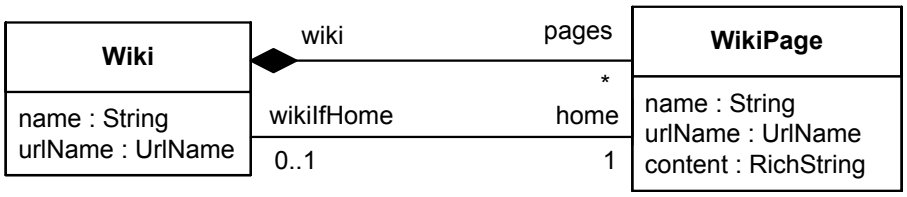

Fig. 2. Overview of an exemplary Tricia data model in a UML-based notation

Tricia data models can be visualized at construction time and at run-time in two different notations:

- A graphical overview notation similar to UML class diagrams (see Figure 2).
- A domain-specific textual syntax which contains all model details (see Figure 3).

The first notation should be self-explanatory and the reader should already get a first idea of the concepts of the DSL used in Figure 3 which are described in the next section.

4 The data modeling concepts of Tricia

As explained in the architecture overview section, the data modeling framework is responsible for the management of persistent and volatile data as specified by the data models of the core and the plugins. Technically speaking, the framework provides (possibly abstract and polymorphic) Java classes for each data modeling concept of Tricia. These classes are instantiated and customized based on the data model of the specific application.

Figure 4 provides an overview of all concepts of the Tricia data modeling framework. In the following subsections we present each of these classes and some of their extension points and illustrate their use with the wiki sample application introduced in the previous section.

4.1 Entities, properties and roles

Entities Tricia domain objects are represented as objects of type `Entity`. The example defines `Wiki` and `WikiPage` entities. Entities have a name, identifying the concept in the data model, and an internationalized label, which is used to generate views for end users. In our example, a `WikiPage` has an internationalized

```
entity Wiki
 label = (en : "Wiki")
 mandatoryMixins
  Linkable
  Seachable
 features
  name : StringProperty
   maxLength = 255
   isIndexed = false
   isPersistent = true
   label = (en : "Name")
   validate
    MinimalLengthValidator(length = 1)
  urlName : UrlNameProperty
   maxLength = 255
   isIndexed = false
   isPersistent = true
   label = (en : "Name in URL")
  pages : ManyRole (WikiPage)
   oppositeRole = wiki : OneRole
   isCascadeDelete = true
   isPersistent = true
  home : OneRole (WikiPage)
   oppositeRole = wikiIfHome : OneRole
   isCascadeDelete = false
   isOwner = true
   isPersistent = true
   label = (en : "Home Page")
```

```
entity WikiPage
 label = (en : "Wiki Page",de : "Wikiseite")
 mandatoryMixins
  Commentable
  Linkable
  Taggable
  Seachable
 features
  name : StringProperty
   maxLength = 255
   isIndexed = false
   isPersistent = true
   label = (en : "Name")
   validate
    MinimalLengthValidator(length = 1)
   onChange
    updateUrlName (
     WikiPage.name,
     WikiPage.urlName
    )
  urlName : UrlNameProperty
   maxLength = 255
   isIndexed = false
   isPersistent = true
   label = (en : "Name in URL")
  content : RichStringProperty
   maxLength = 16777216
   isIndexed = false
   isPersistent = true
   label = (en : "Content")
  wiki : OneRole (Wiki)
   oppositeRole = pages : ManyRole
   isCascadeDelete = false
   isOwner = false
   isPersistent = true
   label = (en : "Wiki")
   validate
    NotNullOneValidator
  wikiIfHome : OneRole (Wiki)
   oppositeRole = home : OneRole
   isCascadeDelete = false
   isOwner = false
   isPersistent = true
```

Fig. 3. Detailed Tricia data model in a textual DSL

label with the English text “Wiki Page” as well as the German text “Wikiseite”. The textual representation of the label is as follows (see also Figure 3)

```
entity WikiPage
  label = (en : "Wiki Page",de : "Wikiseite")
```

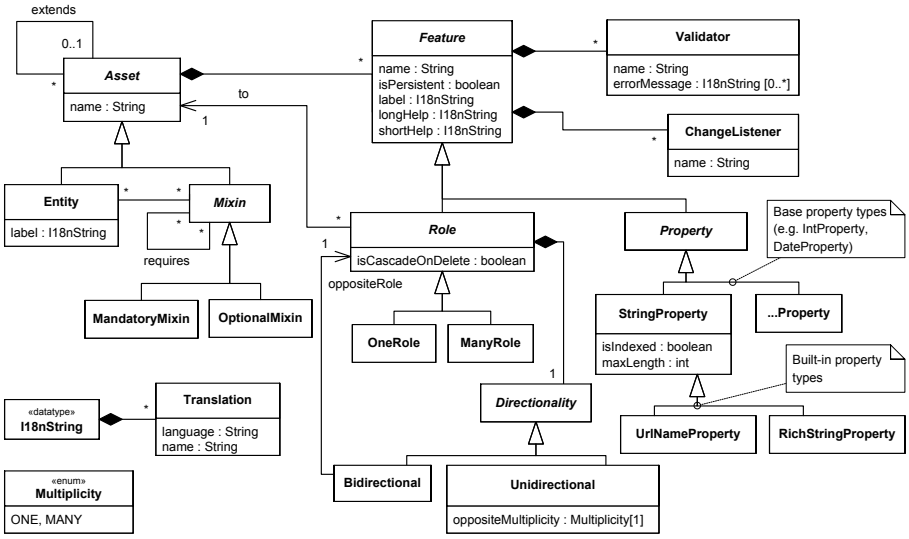

Fig. 4. Concepts of the Tricia data modeling framework

Properties Properties of domain objects are represented as objects of type `Property`. The data modeling framework provides the following predefined basic property types: `BooleanProperty`, `IntProperty`, `StringProperty`, `DomainValueProperty`, `DateProperty`, `TimestampProperty`.
Each property type may introduce certain attributes, which can be customized. For instance, `StringProperty` represents a character sequence, with a size limited by the `maxLength` attribute. The attribute `isIndexed` indicates whether an index shoud be created to speed up value-based queries for that property.

Building on the basic property types (e.g., StringProperty) the Tricia core provides the following domain-specific property types:

- A `RichStringProperty` is a sub type of `StringProperty`, which holds HTML content. The implementation of `RichStringProperty` ensures, that the content does not contain malicious scripts, automatically detects dangling hyperlinks and supports a consistent application-wide URL renaming.
- An `UrlNameProperty` is used to provide meaningful URLs for domain objects. URLs should match as closely as possible the name of the object, but may be subject to additional constraints due to character set limitations for URLs.
- A `PasswordProperty` holds encrypted passwords and makes sure that the content of the property is never displayed in views.
- An `IdProperty` is a sub type of `StringProperty` with the special semantics of being a unique identifier for an entity. Each entity has a property of type `IdProperty`.

In our example of Section 3, properties of a *Wiki* are *name* of type `StringProperty` and *urlName* of type `UrlNameProperty`. A *WikiPage* also has the properties *name*, *urlName*, and additionally a *content* property of type `RichStringProperty`.

Roles Associations between domain objects are represented in Tricia by modeling the association ends as objects of type `Role`. A role specifies the type of the associated entity, which is represented in the data model framework of Figure 4 by the *to* reference.

There are two kinds of multiplicities: A single-valued association is modeled using the class `OneRole` and a multi-valued association through the class `ManyRole`. The directionality of a role is expressed by the mandatory concept `Directionality`. Bidirectional roles reference the corresponding opposite role. In this case the multiplicity of the counterpart is given implicitly through the type of the opposite role instance (`OneRole` or `ManyRole`). The bidirectional `pages` role from the example data model of Figure 2 is textually represented as follows:

```
pages : ManyRole (WikiPage)
  oppositeRole = wiki : OneRole
```

Since an unidirectional role does not specify an opposite role, its multiplicity cannot be derived and has to be defined explicitly through the `otherMultiplicity` attribute.

The attribute `isCascadeOnDelete` indicates to delete the referenced entities if the owning entity is deleted (c.f. UML composition). In our example, a wiki is used as a container for a set of wiki pages:

```
pages : ManyRole (WikiPage)
  isCascadeDelete = true
```

Features Properties and roles share some common attributes, which are captured by the abstract super concept `Feature`. Each feature has a name, an internationalized label (c.f. entity attributes), as well as the two internationalized attributes `longHelp` and `shortHelp`. These labels are used in generated views to describe the meaning of a feature to end users in their own language.

The flag `isPersistent` indicates whether the value of a feature is to be stored persistently in a database, by default this flag is set. A non-persistent property can be used for derived values, which are calculated depending on the values of other persistent properties, and can be shown in certain views. The Tricia data modeling framework also supports inheritance, i.e., a derived entity inherits all features of its parent entity. By default, a single-table strategy [11] is used to map the inheritance tree to a single database table.

4.2 Validators

An important aspect of data modeling is the specification of constraints to ensure data integrity. In the Tricia data modeling framework constraints can be modeled through `Validator`s. As part of the declarative model, a validator has a name and provides error messages, which are shown to end users in case of a validation failure. The actual algorithm, which computes the state of a validator, is provided as a hand-written customization as introduced in section 2. Validators can be specified for all features, i.e., for roles and properties equally.

As an example, a validator verifies whether the value of a `StringProperty` satisfies a specific pattern (e.g., e-mail address). An example for role validation is to constrain the cardinality of an association. In our example, a wiki page has to be part of a wiki. This can be realized by a role validator applied on role *wiki*:

```
wiki : OneRole (Wiki)
 validate
  NotNullOneValidator
```

The Tricia data modeling framework provides a set of built-in property validators, e.g., `EmailValidator`, `MinimalLengthValidator`, as well as predefined role validators, e.g., `NotNullOneValidator`, `NotEmptyManyValidator`. Validators can be parameterized with values:

```
name : StringProperty
 validate
  MinimalLengthValidator(length = 1)
```

4.3 Change listener

In the Tricia data modeling framework `ChangeListener`s are used to propagate data model changes through the system. A change listener has a name and is registered on a feature in order to be notified when the value of the feature changes. Change listeners apply for both kinds of features, i.e., roles and properties equally.

For example, a change listener `updateUrlName` can be defined for the name property (StringProperty) of a WikiPage. If the name property is set for a newly created page and no URL is given by the end user, the value of the `name` property is used as default for the URL. In this case, the URL cannot be empty, this is ensured by the validation rule of the name property (cf. `MinimalLengthValidator` in section 4.2).

4.4 Entities and mixins

The only way of realizing reuse at the data model level introduced so far is the mechanism of inheritance. Since models in Tricia are realized by subclassing

framework classes, this mechanism is constrained by having a single inheritance chain, which means that an entity can have only one entity it inherits from. This imposes a severe limitation, and is not sufficient for real-world modeling problems. To enable reuse on a more fine-grained level, Tricia utilizes the concept of *mixins*[1].

Mixins extend entities with additional properties and roles. As shown in Figure 4, the `Entity` and `Mixin` classes are subtypes of the abstract class `Asset`, which provides the capability of having features as introduced in 4.1. Mixins can be assigned to other entities and vice versa, which is expressed by a many-to-many association between `Entity` and `Mixin` as shown in the class diagram in Figure 4.

We distinguish two kinds of mixins, which are realized by the framework classes `MandatoryMixin` and `OptionalMixin`. Mandatory mixins are assigned statically to a certain entity and cannot be removed at runtime. In Table 1 an extract of existing mandatory mixin types and their use by entity classes is shown. These mixins enable fine-grained re-use.

A mixin can depend on other mixins, e.g., a searchable entity (i.e., an entity the mandatory mixin `Searchable` is assigned to) requires to be linkable (have a URL) too, otherwise the asset cannot be accessed if it is shown in a search result list. In this example, it is not permitted to define searchable entities, which are not linkable.

Table 1. Mandatory mixins and their usage in the core and in the Wiki plugin

	`Linkable`	`Searchable`	`Taggable`	`Commentable`	`Versionable`
`Group`	x	x	x		
`Membership`					
`Person`	x	x	x		
`Principal`	x	x			
`Comment`	x	x			
`Search`	x	x	x		
`Version`	x				
`Wiki`	x	x			
`WikiPage`	x	x	x	x	x

In contrast, optional mixins can be assigned to objects and can be removed at runtime by end users. An example of an optional mixin is the class `CalendarItem`, which can be assigned to wiki pages. It marks the assigned wiki page as representing a temporal event, which is characterized by additional features such as `startDate`, `endDate`, and `eventCategory`. As opposed to mandatory mixins, this capability is not required for all wiki pages, but can be assigned by end users at runtime. The existence of this mixin type then indicates whether a specific wiki page is displayed in a calendar view, or not.

As shown in Table 1, the Tricia core includes predefined entity types which are essential for the domain of enterprise web applications. They comprise entities for

modeling users and user groups: `Person`, `Group`, `Membership`, and `Principal`. These entities are the foundation for the access control framework (see Section 2). Other built-in entites are `Link`, `Comment`, `Version`, which are associated with the respective mandatory mixin types. For example, the mixin `Commentable` establishes a one-to-many association to entities of type `Comment`:

```
mandatoryMixin Commentable
 requires
   Linkable
 features
  showComments : BooleanProperty
   isIndexed = false
   isPersistent = true
   label =  (en : "Show Comments")
  comments : ManyRole (Comment)
   isCascadeDelete = true
   isPersistent = true
   oppositeRole = commentable : OneRole
```

```
entity Comment
 label =  (en : "Comment")
 mandatoryMixins
  Linkable
  Searchable
 features
  authorName : StringProperty
  maxLength = 255
  isIndexed = false
  isPersistent = true
 content : StringProperty
  maxLength = 16777216
  isIndexed = false
  isPersistent = true
  label =  (en : "Content")
  validate
   MinimalLengthValidator(length = 5)
 creationDate : TimestampProperty
  isPersistent = true
 commentable : OneRole (Commentable)
  isCascadeDelete = false
  isOwner = false
  isPersistent = true
  oppositeRole = comments : ManyRole
```

Fig. 5. Comment and Commentable - textual representation

5 Introspective Implementation

As presented in sections 2 and 4, data models are represented in Tricia as Java classes, which instantiate and customize classes of the data modeling framework. In order to enable a model driven development process, declarative models can be extracted from the Java code by *introspection* [4–6]. Technically speaking, the data modeling framework is an introspective whitebox framework, since it provides annotations in the framework classes, which mark the extension points. Customizations have to follow an introspective programming model, which enables the extraction of the model information. For more details see [4–6].

As already mentioned, Tricia provides different views to visualize the data models. The most generic view presents a data model in a tree structure, which is shown in Figure 6. As it is shown in this Figure, the model view is integrated with the Java source it is derived from.

In order to get an overview of a data model, a graphical presentation similar to UML class diagram notation is provided for Tricia application developers. A screenshot showing the wiki data model is depicted in Figure 7. More details on all features in the graphical view are accessible via the textual representation already introduced as shown in the screenshot.

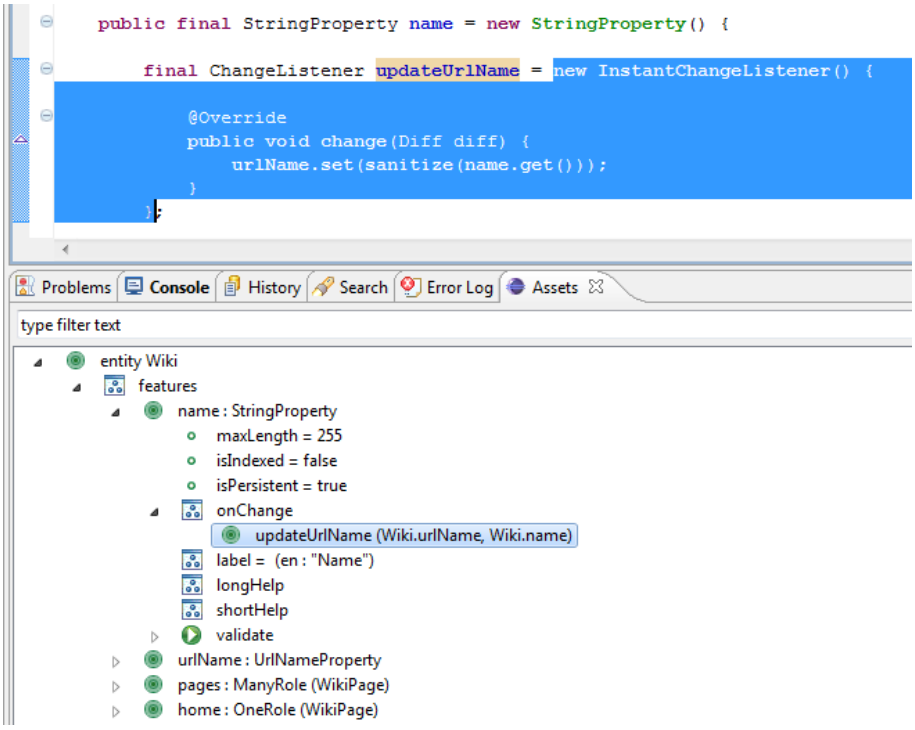

Fig. 6. Tree view of the entity type `Wiki` generated through introspection

6 Related Work

There exist numerous approaches to model driven web development [9, 15–17, 19]. Since the main focus of this paper is on the data model, we will characterize the data modeling capabilities of the following approaches:

– WebML [9] uses a notation which is compatible with classical E/R models and with UML class diagrams. To cope with the requirement of expressing redundant and calculated information, the structural model also offers a simplified, OQL-like query language, by which it is possible to specify derived information.

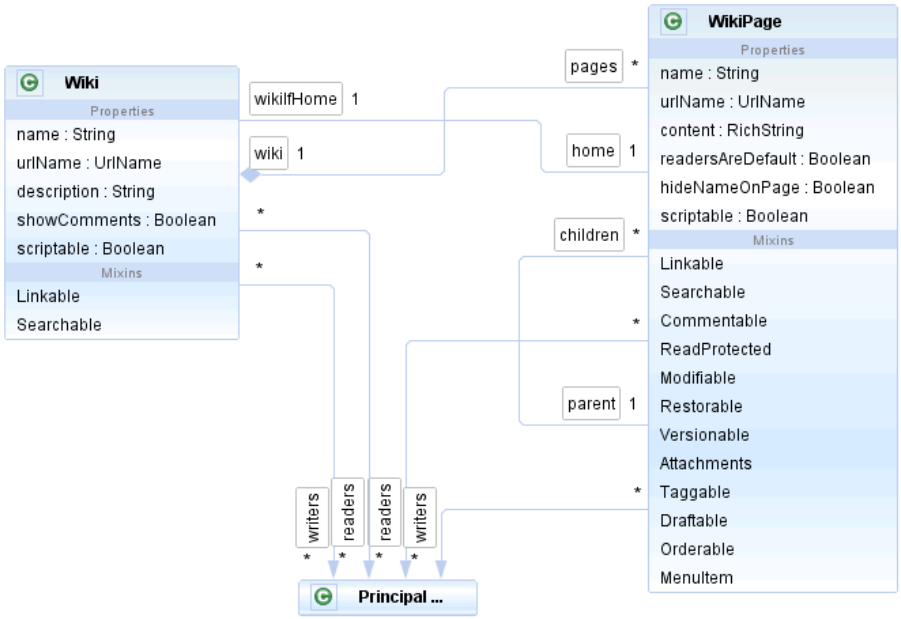

Fig. 7. Introspective Graphical View

- UWE [15] uses the graphical UML class diagram notation for data modeling. The main modeling elements used in the conceptual model are: *class* and *association*. Additional features which can be used to semantically improve data models are: *association* and *role* names, *multiplicities*, different forms of associations supported by the UML like *aggregation*, *inheritance*, *composition* and *association class*.
- Mod4j [17], WebDSL [19], and MontiWeb [16] specify models using a textual representation, which is transformed by a generator into JPA code.

None of the existing approaches supports mixin types, which enable reuse as shown in section 4.4.

These existing solutions all use the *generative* approach to model driven development, which means that source code is *generated* from models. What differentiates our approach from these approaches is the idea to extract models from the source code through introspection, which improves the integration of the models with the underlying system [6].

Our introspective approach is closely related to the one introduced in [2] in the sense that declarative model views are extracted from Java source code. In [2] this idea is being applied to behavioral models.

7 Summary

We presented Tricia, an open source Java-based platform for the development of dynamic data intensive enterprise web applications and social software solutions.

We introduced Tricias architecture, its constituents and interfaces. Tricias plugins enable componentized large applications and provide with mixins the basis for supporting software product lines [18] at the data modeling level. Tricia follows a data model driven approach to system implementation. We gave an example of declarative application development based on the data modeling framework in the domain of social software. Tricia provides compile-time and runtime introspection with a strongly typed generic meta-model. We illustrated how textual and graphical views of introspective models facilitate the understanding of complex web applications.

Based on the proposed architecture we built an Enterprise 2.0 tool, which we compared in [7] to existing commercial and open source tools. Our tool consists of 15 plugins with 40 entities and about 500 features and is used in production in several places (e.g., [10, 14]). Our experiences in building and maintaining a system of this size show that a data model driven approach improves the understandability and quality of the system.

Due to space limitations this paper focuses on the data modeling framework. The access control and interaction modeling framework will be subject of subsequent papers.

References

1. D. Ancona, G. Lagorio, and E. Zucca. Jam - designing a Java extension with mixins. *ACM Trans. Program. Lang. Syst.*, 25(5):641–712, 2003.
2. M. Balz, M. Striewe, and M. Goedicke. Embedding Behavioral Models into Object-Oriented Source Code. In P. Liggesmeyer, G. Engels, J. Münch, J. Dörr, and N. Riegel, editors, *Software Engineering*, volume 143 of *LNI*, pages 51–62. GI, 2009.
3. Bitbucket - Tricia. Website. `http://bitbucket.org/infoasset/tricia-core`; visited on May 30th 2010.
4. T. Büchner. *Introspektive Modellgetriebene Softwareentwicklung.* PhD thesis, Technische Universität München, 2007.
5. T. Büchner and F. Matthes. Introspective Model-Driven Development. In V. Gruhn and F. Oquendo, editors, *EWSA*, volume 4344 of *Lecture Notes in Computer Science*, pages 33–49. Springer, 2006.
6. T. Büchner and F. Matthes. Using Framework Introspection for a Deep Integration of Domain-Specific Models in Java Applications. In *Proceedings of the 1. Workshop des GI-Arbeitskreises Langlebige Softwaresysteme (L2S2): Design for Future - Langlebige Softwaresysteme*, pages 123–135, 2009.
7. T. Büchner, F. Matthes, and C. Neubert. A concept and service based analysis of commercial and open source enterprise 2.0 tools. In K. Liu, editor, *KMIS*, pages 37–45. INSTICC Press, 2009.

8. G. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: a system of patterns*, volume 1. John Wiley and Sons, 1996.
9. S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33(1-6):137 – 157, 2000.
10. ECHORD (European Clearing House for Open Robotics Development). Website. `http://www.echord.info`; visited on May 30th 2010.
11. M. Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003. With contributions from Rice, D., Foemmel, M., Hieatt, E., Mee, R. and Stafford, R.
12. G. Froehlich, H. Hoover, L. Liu, and P. Sorenson. Designing object-oriented frameworks. *University of Alberta, Canada*, 1998.
13. infoAsset. Website. `http://www.infoasset.de`; visited on May 30th 2010.
14. Intranet, Faculty of Informatics, Technical University Munich. Website. `http://intranet.in.tum.de`; visited on May 30th 2010.
15. N. Koch and A. Kraus. The Expressive Power of UML-based Web Engineering. In *Second International Workshop on Web-oriented Software Technology (IWWOST02)*, volume 16. Citeseer, 2002.
16. B. R. M. Dukaczewski, D. Reiss and M. Stein. MontiWeb - Modular Development of Web Information Systems. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM' 09)*, Orlando, Florida, USA, 2009.
17. Mod4j. Website. `http://www.mod4j.org/`; visited on May 30th 2010.
18. K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, Berlin, 2005.
19. E. Visser. WebDSL: A case study in domain-specific language engineering. *Generative and Transformational Techniques in Software Engineering (GTTSE 2007), Lecture Notes in Computer Science. Springer*, pages 2008–039, 2008.