EMF, together with JFace, databinding and persistence implementations, simplifies the development of complex software applications with its modeling mechanisms. However, setting up and assembling all these technologies still requires both deep knowledge of most internal details and a considerable amount of time; the development of applications based on EMF could be simpler, with more code reuse, and without having to deal with too many internal details.

In this talk we present EMF Parsley, a framework that has been recently approved as an Eclipse project. It is based on the previous experimental framework EMF Components [4]. EMF Parsley provides a framework to easily develop UI components based on EMF models. The framework hides most of the complexity of internal details: creating a JFace viewer and connecting it to an EMF resource, usually requires a few lines of code. The main design principle underlying this framework is to split responsibilities into small classes (adhering to the Single Responsibility Principle [9]); thus, customizing a single aspect of the components requires to specialize only the class that deals with that specific aspect, not the whole component. This should maximize code reuse and promote a programming style where the classes implemented by the programmer are usually very small and deal with not too many aspects.

The framework also comes with some UI components that can be used out-of-the-box (including trees, tables and forms, and view and editor parts). Programmers can rely on these components and customize them, or use them as a reference implementation to build their own components based on our framework.

To customize existing components we rely on Dependency Injection [7], in particular using Google Guice. The configuration and setup of Guice modules uses the same mechanism of Xtext (i.e., binding an implementation class only requires to write a specific “bind” method in the main Guice module class). The initial setup of a project which uses EMF Parsley is done via a project wizard, so that the programmer does not have to deal with these details.

Specification of custom behaviors (e.g., label providers, content providers, context menus, etc.) is based on the types of elements of the EMF model, but without requiring long cascades of Java “instanceof” and casts: we provide a polymorphic method dispatch mechanism (borrowed from Xtext) that allows to write cleaner and declarative code.

We also provide a DSL (implemented in Xtext [3, 5]) for making the use of our framework easier: customizations can be specified in a compact form in a single file. The DSL compiler will automatically generate all the Java classes and the corresponding Guice module bindings. The DSL provides a fully feature Eclipse editor and, by relying on Xbase [6], it provides a Java-like language completely integrated with Java and Eclipse JDT.

Setting up views (like in Figure 1) only requires a few lines of Java code. Using the DSL (like in Figure 2) the customization of the components is really quick (the corresponding Java code will be automatically generated by the DSL compiler).

Concerning the persistence aspect, the framework can handle a generic EMF persistence implementation, like XMI or Teneo, just providing the EMF Resource’s URIs. It also includes a bundle for handling CDO resources, which takes care of CDO sessions and transactions transparently.

All sources are covered by tests, both unit tests (with JUnit) and functional tests (with SWTBot). Tests are already integrated in the Continuous Integration system implemented with Jenkins.

Also the building and update site architecture is already setup and implemented with Buckminster [8]. This building tool is heavily used throughout the whole development cycle, starting from the workspace and platform materialization up to headless building and continuous integration.

In addition to be used to render RCP UI, the framework is out-of-the-box ready to be used for RAP development (Remote Application Platform [2]). By using single sourcing techniques, it is easy to develop an RCP application with EMF Parsley and to automatically have a web version (based on RAP) of the same application.

EMF Parsley can also be used with the new Eclipse e4 platform. The framework that is closer to our proposal is EMF Client Platform [1]. However, while the latter aims at providing a quick way of obtaining a full application based on EMF, EMF Parsley aims at providing single smaller and reusable components. Moreover, our components are not customized via extension points, but using injection with plain Java and with a DSL.

The framework presented here is available at http://www.eclipse.org/emf-parsley.

The framework is out-of-the-box ready to be used for RAP development (Remote Application Platform [2]). By using single sourcing techniques, it is easy to develop an RCP application with EMF Parsley and to automatically have a web version (based on RAP) of the same application. The framework that is closer to our proposal is EMF Client Platform [1]. However, while the latter aims at providing a quick way of obtaining a full application based on EMF, EMF Parsley aims at providing single smaller and reusable components. Moreover, our components are not customized via extension points, but using injection with plain Java and with a DSL.

The framework presented here is available at http://www.eclipse.org/emf-parsley.

1. REFERENCES
Figure 1: A Mail RCP Example implemented with EMF Parsley.

Figure 2: Using the EMF Parsley DSL for customizations.