AIE SDK Overview

ABAP in Eclipse Software Development Kit Overview

ABAP in Eclipse offers state-of-the-art ABAP development tools on top of the Eclipse platform combining ABAP's strong and proven lifecycle management with Eclipse's UI and integration capabilities. The goal of the ABAP in Eclipse Software Development Kit (SDK) is to enable other teams to enhance the ABAP development experience by contributing their own tools. This paper provides an overview of the architecture, with a special focus on the integration of new tools using this SDK.

Introduction

Motivation

One reason to choose Eclipse as the platform for the new ABAP development environment is its openness towards new tools and features via the Eclipse plug-in Framework. The new ABAP development infrastructure provides these plug-in capabilities also on the ABAP backend. That means, by making use of Eclipse plug-in capabilities on the frontend and ABAP development infrastructure plug-in capabilities on the backend, it is possible to contribute to ABAP development environment with new development tools. Those tools will provide the well-known ABAP capabilities (transport, activation, version management, "Where-Used" list etc.) combined with Eclipse-based state-of-the-art UI and integration capabilities in a new highly integrated development environment.

Big Picture

The **Eclipse Client** offers a modern UI with fast client-side rendering, multi-threading, and multiple views in the same context. The **Eclipse Client** can be used to connect with multiple **ABAP backend** systems at the same time, even of different versions. As usual in ABAP, development takes place online, i.e. directly on the ABAP backend system. As one consequence you do not have to synchronize the content of whole backend systems to your local system to search and navigate to any development object.

By means of service-enabling of the ABAP development infrastructure, the ABAP backend offers a set of REST services (**Re**presentational **S**tate **T**ransfer, see REST at Wikipedia). Developers can use them to build compelling tools for the ABAP development environment. The services themselves are language and platform independent and are designed for easy consumption by any type of client (not only ABAP in Eclipse). Clients communicate with the server via stateless or stateful RFC connections. The communication protocol is HTTP (with some minor deviations).

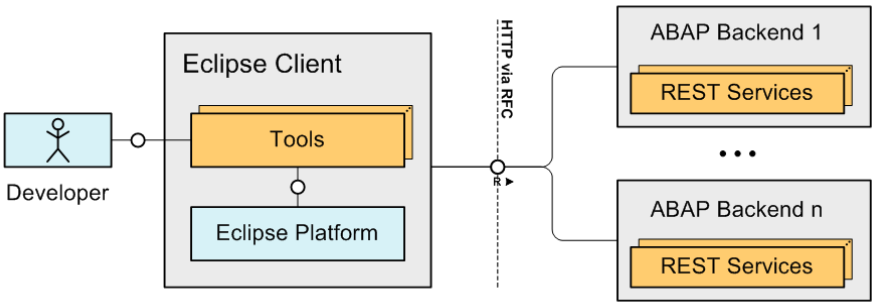


Figure 1: ABAP in Eclipse Client connecting to multiple ABAP Backends

Architecture Overview

Now, let's take a more detailed view at the different components involved.

ABAP Backend tools and frameworks

The core of the **Development Infrastructure (DI)** in the backend is built up of an open set of development tools and their corresponding frameworks like **SE80** / **ABAP Workbench**, **SE09** / **Change and Transport System** (CTS), **Repository Infosystem** (RIS), Version Management, **Debugger**, etc. (See Figure 2).

Development Infrastructure Services

One of the central challenges of the new ABAP in Eclipse development infrastructure consists of providing a clean **DI service** layer (without UI) for the above mentioned tools and frameworks, so that they can be used by remote tools.

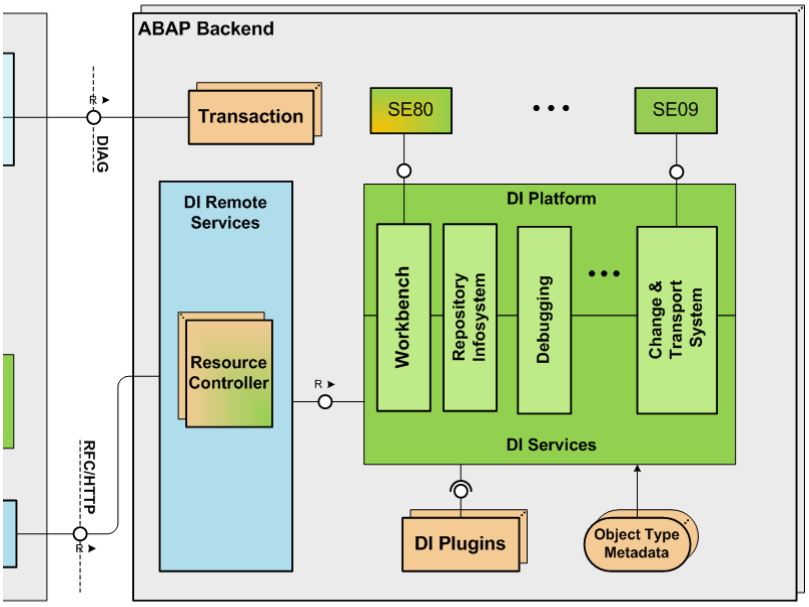


Figure 2: Backend Architecture of the ABAP Development Infrastructure

Backend extensibility

The majority of existing Workbench tools like Class Builder, ABAP Editor and Function Builder, ABAP Dictionary (SE11), etc. are built-in into the Workbench. The Development Infrastructure provides different plug-in mechanisms to enable easy integration of new development object types. These are created by providing domain specific metadata (**Object Type Metadata**), and implementations for the integration and tool logic (**DI Plugins**). Note that in the case of the ABAP Workbench, the plug-in mechanism is a rather new feature (available since NetWeaver release 7.31).

In addition to new workbench object types, it is also possible to create new DI Services as well as to provide service-enabling of existing tools and frameworks such as: Debugger, ABAP Test Cockpit, SAT Runtime Analysis, etc.

Remote API

With simplicity and easy consumption for any client in mind, **DI Remote Services** provide REST-full service APIs for the integration of development tools. This will facilitate and significantly speed up the later integration of any tools into ABAP in Eclipse and potentially any other client. Note that the responsibility of this layer is "only" to wrap the DI Services API (mostly stateful and table-oriented) to a REST API (mostly stateless and XML-oriented).

The DI Remote Services provide out-of-the-box support for some common backend functionality. Generic **Resource Controllers** enable browsing the system content, provide activation services, search and where-used list, etc. Domain-specific **Resource Controllers** for your new object types can be easily implemented by inheriting from the base classes, which provide basic integration into **DI Services**. REST architectures Uniform Resource Identifiers (URIs) are used to address and identify resources, i.e. development artifacts, e.g. ABAP classes. A pluggable URI mapping service (not shown in the picture) is used to map URIs into workbench requests (object name, object type) and vice versa.

Eclipse Client

As the Eclipse platform (including its tools and services) is based on a local workspace concept built on top of a local file system, a coupling is required between remote resources and local files. To avoid the need of synchronizing large amounts of artifacts to the client, we use a virtual file system called **Semantic File System (SFS)**. This technology enables on-the-fly coupling of remote resources with local files. This way, resources can be loaded on demand.

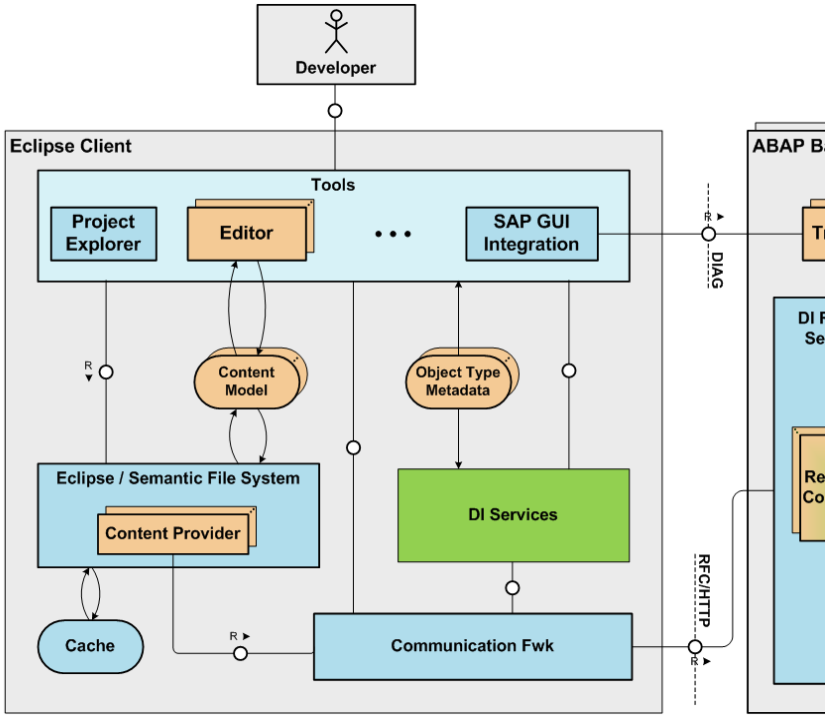


Figure 3: Client Architecture of the ABAP Development Infrastructure

For the integration of a concrete backend repository, the SFS offers a service-provider interface with which so-called SFS **Content Providers** can be plugged-in (See Figure 3). The AiE framework provides generic base classes for such Content Providers which can be extended. In the simplest case, the domain-specific content provider functionality can be defined declaratively via (domain specific) **Object Type Metadata** in the client.

Beside the Eclipse file-to-backend resource URI mapping information, the **Object Type Metadata** consists of labels, icons, actions and other declarative information specific to an object type that is used by the **DI Services** and different generic tools.

Once remote resources are available as local files, a domain specific **Content Model** is used by the Eclipse tools to read and manipulate the in-memory representation of the resource content. For object types that use an XML representation, we recommend using the **Eclipse Modeling Framework (EMF)** (See EMF Project in Eclipse).

Eclipse Tools

Multiple **Tools** build up the UI layer of ABAP in Eclipse. They comprise a rich set of editors, views, wizards, dialogs, etc. Some of these UI components are Eclipse standard, for example the Problems View; others are generic for ABAP development objects, such as the extensible **Project Explorer** View, the Open ABAP Development Object Dialog, etc.

Natively supporting an object type requires providing at least one domain-specific **Editor** and, optionally, a Creation Wizard, customized Properties View, etc. Object types, for which there is still no native editor support, can be edited in the AiE environment by means of the **embedded SAP GUI**. This can be also used to run any Dynpro transaction in the client.

Client-backend compatibility

To ensure backward compatibility of the ABAP in Eclipse client (there is only one client version that must work with any enabled backend version), **Tools** use the ADT compatibility and feature discovery framework (part of the **DI Services**) to determine which features are available in a given ABAP backend system.

Extending ABAP in Eclipse

ABAP in Eclipse extensions range from client enhancements only, like for example providing a graphical viewer for an existing editor, to new Workbench object types, including backend integration, remote API, Eclipse editor, wizards, etc. After the architecture overview was presented in the previous section, this section will briefly summarize which elements must be provided to contribute to ABAP in Eclipse. For this purpose, we use an extension scenario including backend and client elements for a new Workbench object type:

Define Object Type and provide required plug-ins

The creation of new Workbench object types starts with defining a new Logical Transport Object in CTS (transaction SOBJ). After this, the integration into the development infrastructure frameworks (Workbench, RIS, Versioning, etc.) can be declared using the Object Type Editor, where you provide general information about the new object type, and declare the ABAP classes which build-up your backend plug-ins.

Provide remote API

Using available base classes, the implementation of a resource controller for the new object type can be almost reduced to provide a content handler, responsible for creating and consuming the content of the REST resource in XML form. The next steps consist of registering the new resource controller and to provide the mapping between the new URIs, which identify the new Resource types, and the new Workbench object type.

Note that the contract between backend and client is defined in this step and consists of:

- The URI layout, which defines how the address of the new Resources looks like, and
- The XSD, which defines the structure of the resource content.

Define client-side metadata for new object type

At the client side, information must be provided about the URI and the file layout to be used for the REST Resources of the new object type. Also icons and other information can be provided using Eclipse extension points in the plug-in definition.

Create content metamodel

For the XSD, which defines the content format of the Resources, an ECore metamodel and the corresponding EMF Java classes can be automatically generated (See EMF Project in Eclipse and EMF Tutorial).

Implement native Eclipse editor

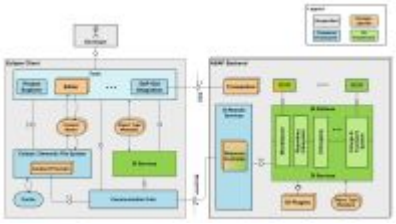
An Eclipse editor for the new object type can be easily implemented using basis classes which provide the necessary integration with the Semantic File System, Locking and Transport Selection dialog. The implementation task consists of providing the UI controls and the data bindings between the EMF model and the UI.

Add native Eclipse wizard for new objects

Finally, a wizard can be provided to create new development objects of the new object type.

Summary and next steps

This document presented an overview of the ABAP in Eclipse SDK architecture, with a special focus on the elements that must be provided by tool developers in order to contribute new tools. See the complete overview picture:



To get a deeper understanding about how to create new object types and contribute new ABAP in Eclipse editors, refer to the end-to-end "Blue Banana" example: tutorial for server-side parts, tutorial for client-side parts