Visual Studio .NET DCL Add-In

— User Guide —

Hasso Plattner Institut for Software Systems Engineering
Chair for Middleware and Operation Systems

Alexander Klimetschek
alexander.klimetschek@hpi.uni-potsdam.de

Alexander Saar
alexander.saar@hpi.uni-potsdam.de

Marc Assmann
marc.assmann@hpi.uni-potsdam.de

Hasso Plattner Institut for Software Systems Engineering GmbH
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam
P.O. Box 900460, 14440 Potsdam

Contents

1 Introduction

The guide at hand introduces the architecture, implementation and usage of the *Distributed Control Lab* (DCL) Add-In for *Visual Studio .Net*. The add-in enables the creation, execution and management of DCL jobs. It was implemented during a project of the ***DIS****tributed & **CO**llaborative **U**niversity **R**esearch & **S**tudy **E**nvironment* (DISCOURSE, [3]) block lecture 2004.

1.1 Distributed Control Lab (DCL)

The DCL [2] is situated at the *Operating Systems & Middleware Chair* [1] of the *Hasso Plattner Institute* (HPI, [5]) and deals with software paradigms and design patterns that allow an interconnection of middleware-based components and embedded mobile systems. The primary point of interest is how to reach a predictable system behaviour (regarding to timing behaviour, fault tolerance or resource usage) in an unstable environment. The evaluation of the different approaches is done with the help of case studies, one example is a web-based control of mobile robots in the lab.

Publishing experiments over the web causes problems that deal especially with non-functional application properties such as fault tolerance, security or realtime. Several methods are developed to deal with these problems. One possible solution is the usage of dynamic reconfiguration as a safe-guard mechanism for user code downloaded from the Internet. Damage to the experiment can be avoided. A configuration framework with mechanism for dynamic reconfiguration are also developed.

2 User Guide

This section focuses on the use of the DCL Add-In for Visual Studio .Net 2003. Users should be familiar with the DCL project and concepts before starting here.

To work with the Add-In, an account for the DCL lab is necessary.

2.1 Installation

The DCL Add-In for Visual Studio .Net 2003 is distributed with an MSI-installer. There is only one additional manual step necessary to complete the installation: Please run

```
$VS_HOME\Common7\IDE\devenv.exe /setup
```

from command line to register the add-in icons in the Visual Studio .Net environment.

2.2 Tool Bar and Code Editor Context Menu

Figure 1: DCL Add-In Toolbar

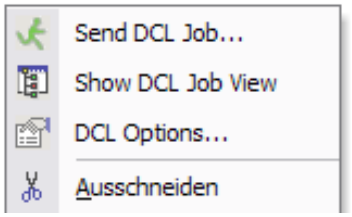


Figure 2: DCL Add-In Context Menu

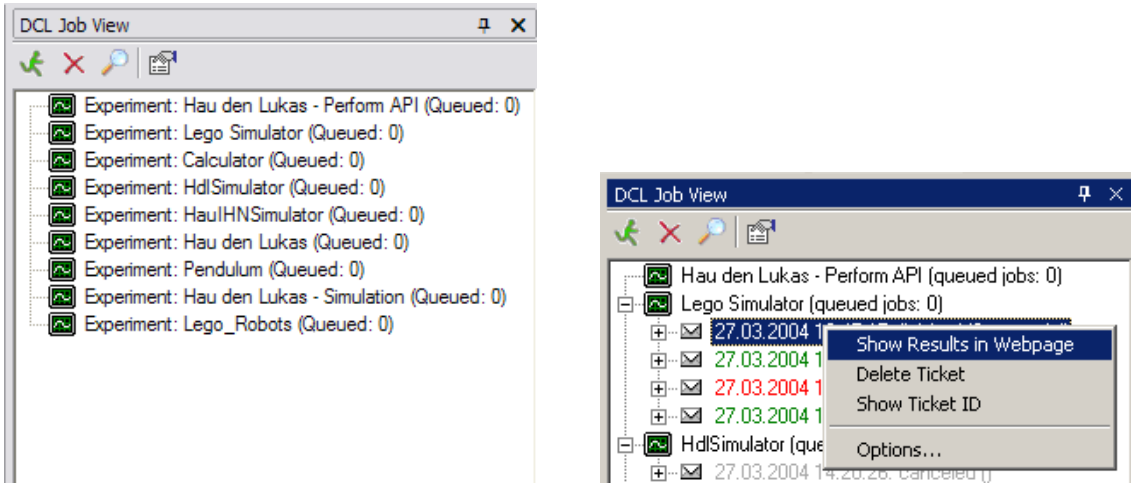
The DCL tool bar as well as the code editor context menu allow to access to the Add-In's features.

Send Job When a source file is opened in Visual Studio, this opens a dialog to choose an experiment from the DCL to send the source file to. After choosing an experiment, the source file is is submitted as a new DCL job. To enable observation of this new job, the Job View is opened and the new job is highlighted yellow.

DCL Job View Opens the DCL Job View.

DCL options dialog Opens the options dialog.

2.3 DCL Job View



(a) Overview (b) Context Menu of Jobs

Figure 3: DCL Add-In Job View

The DCL Job View allows to manage all jobs a user sent to DCL. The control lists all accessible DCL experiment types in a tree view. For every experiment type, the current job-queue length at the DCL lab is displayed. The context menu of an experiment allows to submit the currently opened document in the code editor as a new job.

Below experiment types, all previously sent jobs of a user are listed, as long as they are still registered with a ticket from the DCL ticketing service. Jobs are sorted by submission time and their status is displayed. Below a job, various aspects can be accessed:

- The source code that was submitted to start a job can be opened in read-only mode in the code editor.
- The console output of the job after execution can be opened in the code editor.
- Various job results available at the website. A double click opens a browser to display these results like state graphs, flash movies, etc.

The jobs context menu allows the following:

Show Results in Webpage Opens the results or status of a job in a browser within Visual Studio .Net

Cancel Job Only available before the execution of a job is finished.

Delete Ticket Only available after the execution of a job is finished. Deletes a jobs ticket. This removes all information about a job from server and Job View.

Show Ticket ID Displays the GUID that a job is registered with at the server.

2.4 Options

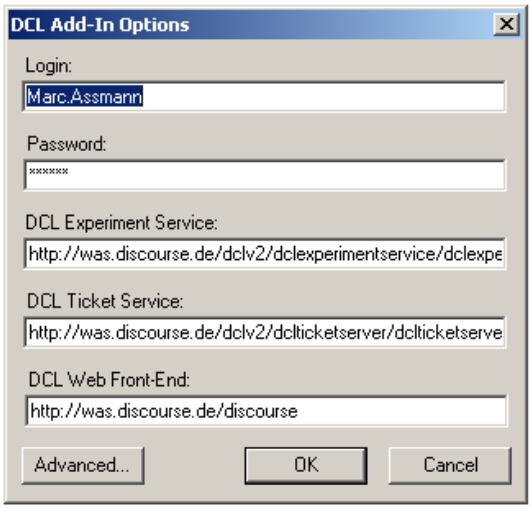


Figure 4: DCL Add-In Options Window

The options window allows to enter account information (user and password). The URLs to

- DCL Experiment service
- DCL Ticket service
- DCL Web Frontend

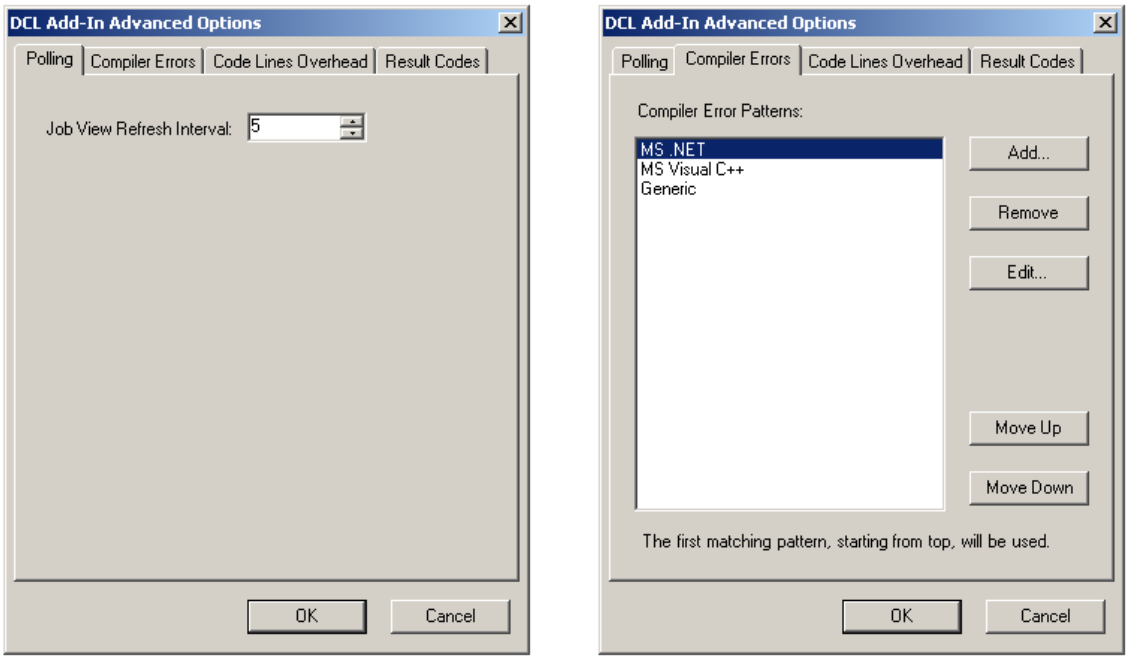
can be adjusted, but this is mostly not necessary as these values default to the DCL installation at HPI. On connecting to the web server, the Web Frontend URL will be suffixed with `/jobdetails.aspx?ticket=TICKET` where TICKET will be replaced with the ticket GUID string (if this is subject of change, the source code must be modified).

2.4.1 Polling Options

The Add-In requests the status about experiments and jobs in intervals that can be set here. To reduce traffic, it can be set to a higher value then the default of 5. Setting it to a lower value is not recommended as this increases traffic a lot.

2.4.2 Compiler Errors Options

To fill the task items with error messages containing file and line number information, compiler output must be parsed. These errors are recognized by regular expressions that can be adjusted



(a) Polling Options

(b) Compiler Errors Options

Figure 5: Advanced Options Part One

here. The error patterns are tried in the same order as they are listed. You can change the order with the ”Move Up” and ”Move Down” buttons at the bottom left.

Adjusting these settings should only be necessary when experiments with other languages are added to DCL.

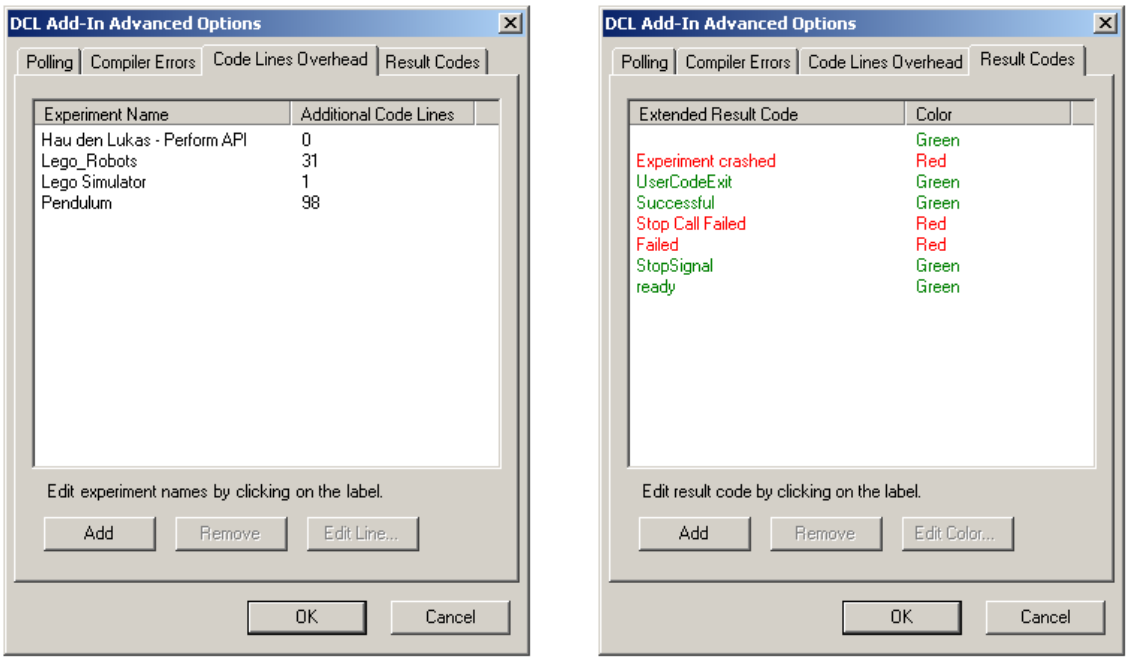
2.4.3 Code Lines Overhead Options

This settings only need to be adjusted when new experiment types are added to the DCL or the line numbering information in task items is incorrect.

When submitting a job to the DCL the source code is embedded into code templates at the DCL server. Therefore, information about line numbers in the compiler output is not correct for the submitted code. To display correct line numbering information in task items, the code lines overhead must be adjusted correct.

2.4.4 Result Codes Options

The Result Codes options allow to customize the coloring of job display for different kind of status codes.



(a) Code Lines Overhead Options

(b) Result Codes Options

Figure 6: Advanced Options Part Two

2.5 Task Window

The task window displays tasks that are generated from compilation errors in jobs that were submitted to the server. This allows to easily track down source errors as with source code compiled and executed on the local system: A double-click on a task item opens the failed jobs code in the editor and places the cursor in the line an error was encountered. If the last sent job (the one that is highlighted yellow) generates compilation errors, they will be displayed automatically in the task window.

2.6 Message Window

The message window displays a log of the Add-In. Start of new jobs is logged here.

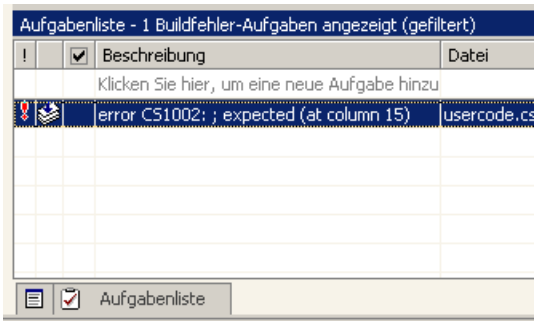


Figure 7: Task Window

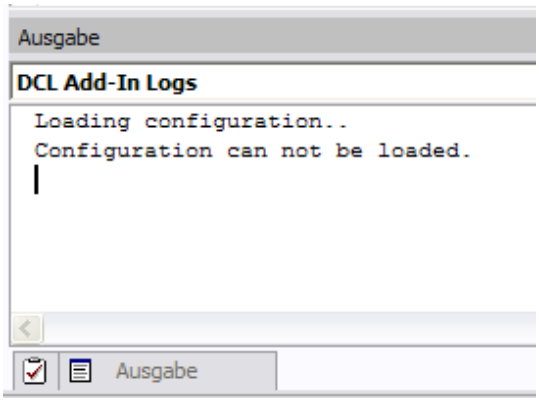


Figure 8: DCL Add-In Message Window

3 Developer Guide

This section shows an overview on the considerations and steps that has been carried out within the development process of this project.

3.1 DCL Web-Service Front-end Control

The following list describes the identified requirements for the Add-In, in other words what should the add-in be able to do.

1. The add-in have to provide a configuration. The configuration should include:
    - user name
    - password
    - URL of DCL ticket service
    - URL of DCL experiment service

- URL of DCL web front-end

2. The add-in have to enable the remote controlling of the DCL. This includes the
   - creation
   - execution
   - cancellation and
   - deletion

   of DCL jobs.

3. The add-in have to enable the display of DCL job results like
   - compiler output
   - flash movies
   - diagrams or
   - job code.

3.1.1 DCL Web-Service Interface

The DCL provides a Web-Service API that can be used to interact with it over a simple and cross-platform available interface. The DCL Web-Service interface consists of two main components. On the one hand the ticket service provides basic functionality for user authentication. This means that valid users can acquire ticket from the ticket service. This tickets are used to create new jobs and to find all jobs of a specific user.

On the other hand the experiment service gives an API for the creation, execution and management of DCL jobs. The tickets acquired from the ticket service are necessary work with the experiments service, because every job is assigned and identified by a ticket.

Both services are implemented in ASP.NET and at the time they are available in version 2. For more informations about the service APIs take a look at [2] or at the folder dcl-doku at the root of the CD.

3.2 Implementation Overview

The core of the add-in was designed as a .NET user control. This control manages the web-service calls and the visualization of the DCL experiments, jobs and results. These are displayed in a tree where jobs are child nodes of experiments and results child nodes of jobs. The option forms are integrated into the user control.

3.2.1 Observer Pattern with Events and Delegates

To use the control in an application it is necessary to get notified about events in the control, e.g. a double-click on a result node. This was implemented with the observer pattern [4], by using C# events and delegates. The following events are available:

LogMessage Indicates when a logging message occurs.

SendJobClicked Indicates when the "Send Job" button in the toolbar of the control or in the context menu of an experiment node has been clicked.

ShowStringResultClicked Indicates when the "Show Result" menu item in a result context menu has been clicked and the result is a string result.

ShowCompilerResultClicked Indicates when the "Show Result" menu item in a result context menu has been clicked and the result is a compiler result.

ShowBinaryResultClicked Indicates when the "Show Result" menu item in a result context menu has been clicked and the result is a binary result like diagrams or flash movies.

ShowWebResultClicked Indicates when the "Show Results in Webpage" button in the toolbar menu item in a result context menu has been clicked.

3.2.2 Static Structure

The diagram in figure 9 shows an overview of the main classes. These are `JobViewControl` itself as well as configuration classes (`Configuration` and `ConfigurationManager`), option form classes (`OptionsForm` and `AdvancedOptionsForm`), as well as the `ExperimentSelectionForm` which is displayed on sending a job.

The most important public methods of `JobViewControl` are listed below. These methods enable the use of the control in a programmatical manner.

startUpdateProcess() Starts the background thread that regularly ret rieves the jobs of the user and updates the tree view accordingly.

stopUpdateProcess() Stops the background thread.

sendJob() Allows to send a job as code in a single string to be sent to the DCL. This method is overloaded and has two variants: One to explicitly name the experiment and another one which takes the currently selected experiment in the tree view or otherwise pops up an `ExperimentSelectionForm`.

showOptions() Shows the `OptionsForm`.

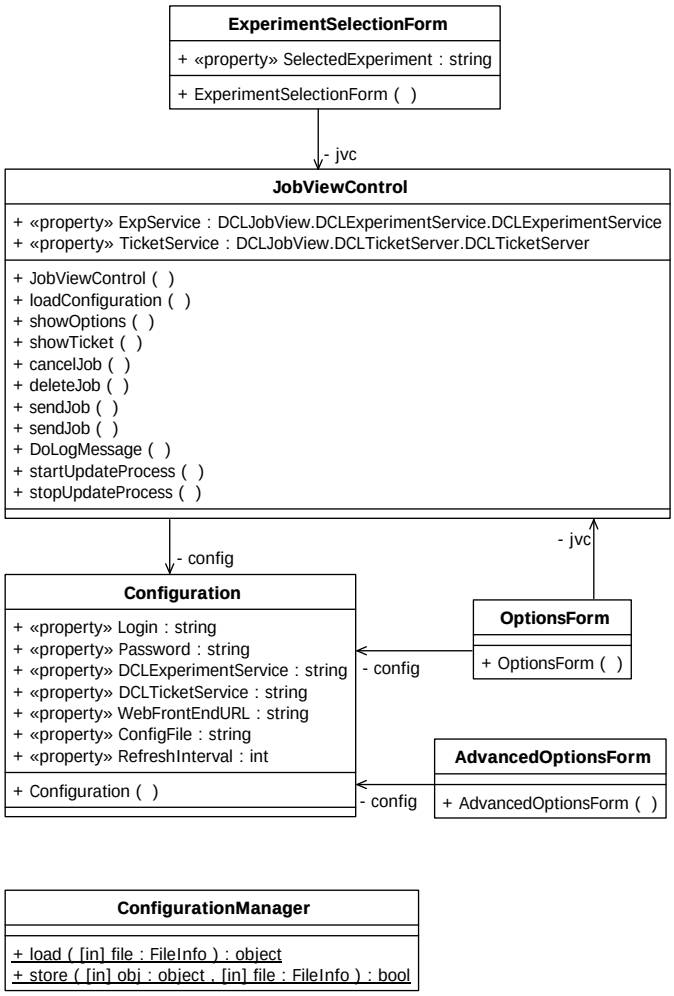


Figure 9: JobViewControl Overview

3.2.3 Configuration and Logging

The `ConfigurationManager` is used to load and store the user specific add-in configuration in the users home directory which is specified by the environment variable %HOME%. The configuration is serialized by the .Net SOAP formatter. As opposed to that the handling of log messages is delegated to the application that uses the control by a specific event.

3.2.4 Tree Model

For the visualization of DCL experiments, jobs and results a tree view with modified nodes was used. An overview of the classes used in the tree view is illustrated in figure 10.

The tree view is regularly updated by a background thread. We decided not to rebuild the entire tree from the scratch every time. The problem with this approach would be that the state of expanded nodes would be lost and an annoying flicker would occur. Thus our implementation includes an intelligent update mechanism which is partly visible in the UML sequence diagram in figure 11.

The first step is to update the experiments, which are the top level nodes. This is done by retrieving the list of experiments. This list is passed to every experiment node (HandleExperimentList()) which has to take care of either updating itself or removing itself from the tree view if it cannot find its name in that list. After that it removes its name from the list to ensure that all expriments left in the list after that process are new experiments. These are added as new experiment nodes.

The updating of jobs works in the same way. The list of tickets is retrieved and then passed to each job (HandleTicketList()). The experiment node class and the job node class both inherit from the JobViewTreeNode class which defines the HandleTicketList() method. The implementation in the experiment node simply passes it to all its child job nodes, the imlementation in job node takes care of the updating.

Depending on the status of the job the Update() method looks for results belonging to this job and updates them in the same way as experiments and jobs are updated. The result node class contains a method HandleResultList() accordingly.

All types of nodes encapsulate information about their specific properties, for example job status, job extended result code or result type (string or binary).

3.2.5 Parsing Compiler Errors

Compiler errors are parsed with regular expressions that are tried line by line on the compiler output. There are three kinds: error, warning and extra line. An error regexp for a certain compiler should match for lines with an error, a warning regexp for warnings and the extra line regexp is tried at the lines following a successful match of one of the three kinds (some compilers pass additionial information on a second line). Error patterns can be fully configured. They are tried in the order of their priority.

The implementation of this was inspired from the Console plugin of the java-based text editor jEdit (`http://www.jedit.org`), released under the GNU General Public License (GPL).

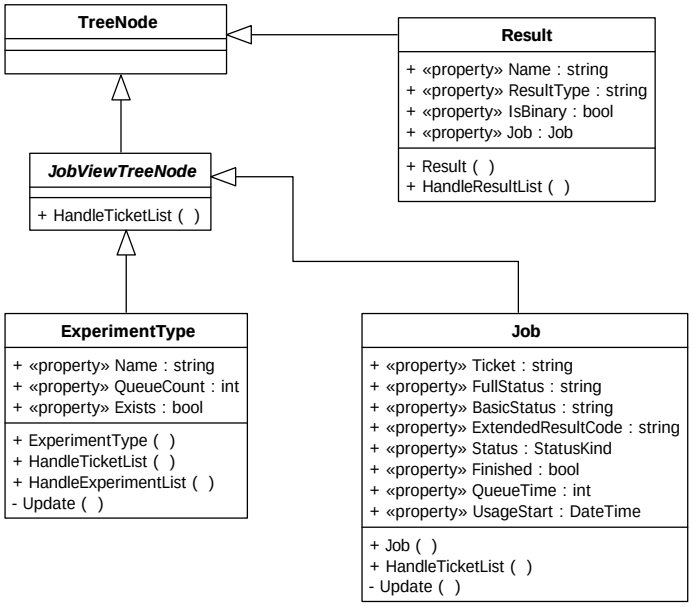


Figure 10: DCL Add-In Tree Model Overview

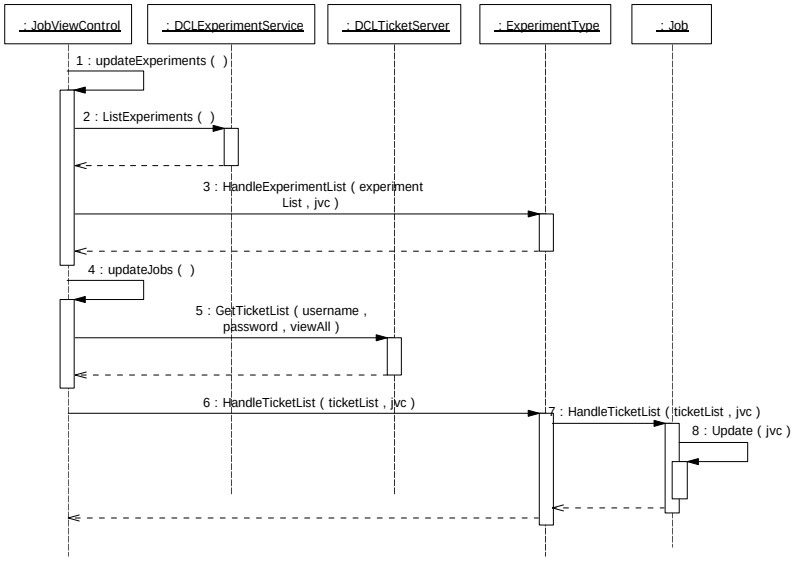


Figure 11: DCL Add-In Tree Model Update

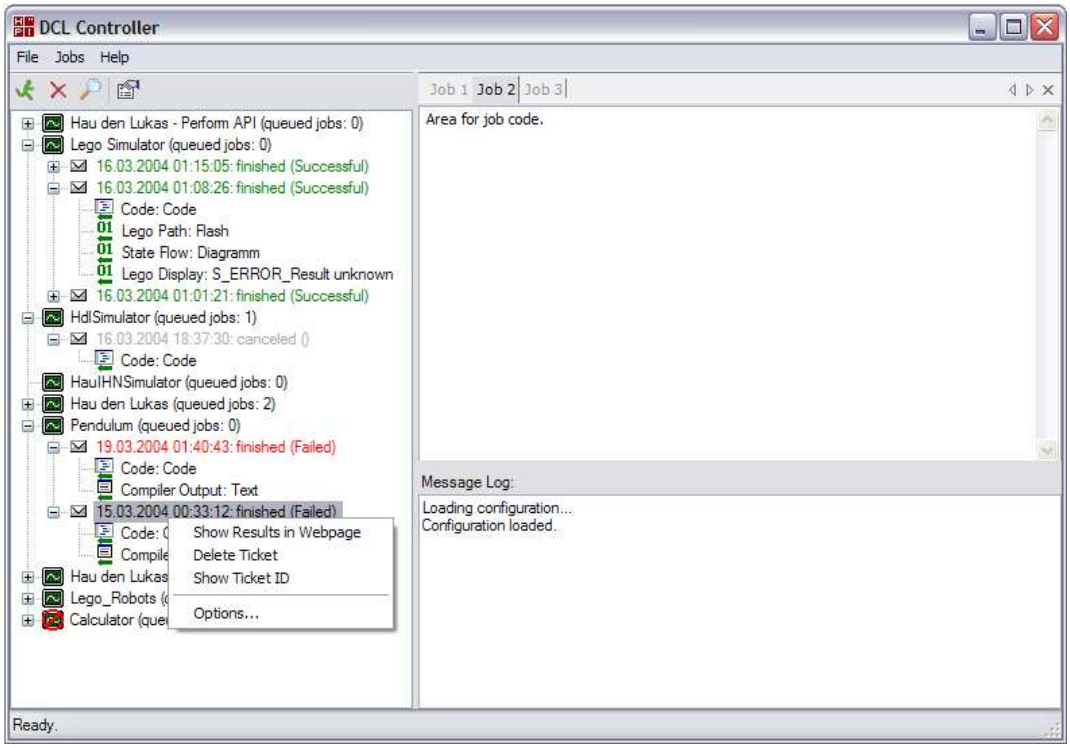


Figure 12: Standalone DCL Front-End

3.2.6 Usage of the DCL Control

As shown in the previous section the of the add-in was focused on one main component, the job view and control component. The advantages of the approach is the re-usability of the control in every type of application that supports Windows Forms or ActiveX components. Additional advantages are encapsulation of the core functionality and the easy usage in the add-in with the *CreateToolWindow()* method, provided by the Visual Studio extension API.

The main important disadvantage of this approach is the higher programming effort necessary for the implementation of the control.

To show the re-usability of the control, a standalone DCL front-end application that uses the control was also part of the project. This application, in addition to the add-in implementation, can be used to understand the usage of the control. A screenshot of the standalone front-end is shown in figure 12.

The usage of the control (in the add-in) is illustrated in figure 13.

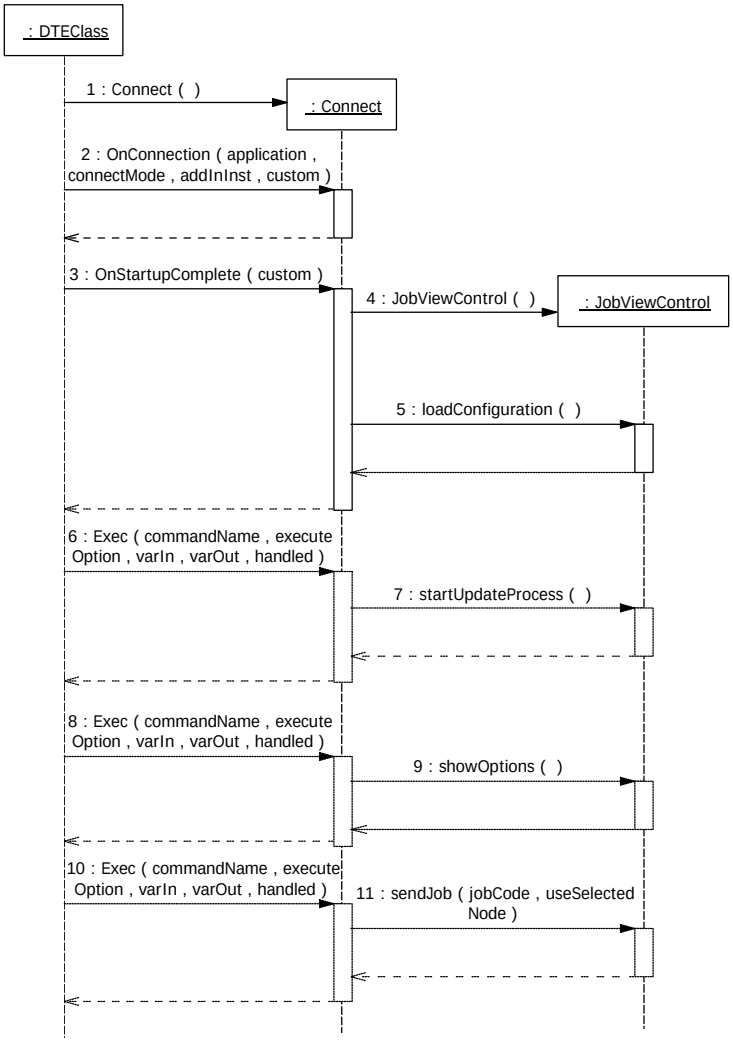


Figure 13: JobViewControl Usage

3.3 Visual Studio .NET Add-In Integration

3.3.1 Add-In Registration

An add-in is a COM component which implements the IDTExtensibility2 interface. Visual Studio .NET looks for add-ins in the registry under ”Software/VisualStudio/7.1/AddIns”. The version number is important, only add-ins installed in the same registry key as Visual Studio are loaded. If there will be a newer version of the IDE, the version key must be changed in the installer (DCLAddinSetup project).

Each add-in has a subkey which is named after the ProgID of the add-in COM component. There can be several values in this key with LoadBehavior (DWORD) as the minimum requirement. It must have a value of 1 initially (this value is modified by Visual Studio). CommandPreload (DWORD) should be set to 1 to automatically start the add-in. FriendlyName, Description and AboutBoxDetails are strings that should describe the add-in, these are displayed in the add-in manager and in the about box of Visual Studio.

3.3.2 Connect Class Overview

As noted, the entry point for an add-in is a COM component that implements the IDTExtensibility2 interface. This is done by the Connect class of the add-in. Additionally it should implement the IDTCommandTarget interface to get triggered for the execution of actions. Figure 14 shows a static structure of the classes in the add-in.

There are four important methods:

OnConnection Called when loading the add-in. This can happen in a different context as identified by the connectMode parameter. In this method we create all the user interface elements and register for certain events to interact with the task list and text documents. (Part of the IDTExtensibility2 interface).

OnDisconnection Called when the add-in is unloaded. (Part of the IDTExtensibility2 interface).

QueryStatus Asks for the availability of a command which is identfied by name. This can be used to set custom commands (that are accessible through a toolbar for example) as enabled or disabled. (Part of the IDTCommandTarget interface).

Exec Called when a custom command belonging to this add-in should be executed. In this method the three commands execute job, show DCL tool window and show option window are implemented. (Part of the IDTCommandTarget interface).

Most parts of the class control the JobViewControl in the tool window. The UML sequence diagram in figure 13 shows the calls made.

3.3.3 Integration into the User Interface

The add-in is accessible throughout four user interface elements inside Visual Studio. There is a standard MS Office toolbar with three commands. The same commands are also available in

Figure 14: DCL Add-In Overview

the context menu of text document windows. The main interface is found in a separate dockable tool window. Finally there is a DCLAddin pane in the standard output tool window.

To add buttons to toolbars it is necessary to create `Command` objects. These need to have a name (which can be also used in Visual Studio's command window), contain an icon and can be reused multiple times in different `CommandBar`s (toolbars and menus).

The removal of these UI components is necessary on installation. Though `Command`s for non-existing add-ins are automatically removed by Visual Studio, toolbars are not. This requires a custom uninstaller to be run during uninstalling the add-in, which opens a Visual Studio instance via COM and removes the toolbar (along with the commands).

A logging window is created by adding a new OutputWindowPane to the output window object.

The registration of a tool window is a bit difficult. There is a method CreateToolWindow() which accepts the ProgID of an ActiveX component which is displayed inside the window (in our case this is the JobViewControl). For convenience, the position of the tool window should be saved between two runs of the IDE. The CreateToolWindow() method must be called on each startup, what requires Visual Studio to recognize an earlier registered tool window with the same custom GUID. Normally the first call to that method fails with an NotImplementedException. A second call (in the catch block) will work but now the position and dock status of the window is lost. The exception can be avoided with a "hack" (found at the beginning of the RegisterToolWindow() method in the Connect class).

To be able to open new windows for the code of older messages as well as recognizing already opened windows the Connect class keeps track of the windows opened. A hashtable includes a mapping from tickets to the windows showing the code of the ticket. This requires to listen for the WindowClosing and WindowActivated events.

Clicking on compiler errors in the task window opens the window with the code and positions the cursor at the point of the error or warning. That requires to listen for the TaskNavigated event.

3.3.4 Serialization

Using .NET serialization in an add-in requires the assembly with the serializable classes to be put into the "VS_NET_INSTALL_DIR\Common7\IDE\PublicAssemblies" folder. On deserialization the .NET framework looks for the assembly of the class which is stored in the serialized data stream. Because an add-in is just a COM component, the main application does not know about the exact locations of the .NET assemblies behind it. Visual Studio looks into the PrivateAssemblies and PublicAssemblies folders for .NET assemblies.

3.3.5 More Information

For more information about Visual Studio Add-In development we found the following resources very useful:

- The yahoo group *vsnetaddin* (a mailing list).
  `http://groups.yahoo.com/group/vsnetaddin`

- `http://www.knowdotnet.com`
- MSDN online. `http://msdn.microsoft.com`

More links can be found in the folder vsaddin-doku at the root of the CD.

3.4 Building the Solution and Projects

Before you build or work on the sources, please add the file DCLJobViewControl/res/ MagicLibrary.dll (contains Cronwood controls) to the toolbox. The DCLController MainForm uses the TabControl found in this assembly.

The DCLAddin solution contains the following seven projects:

DCLAddin The actual add-in which realizes the integration into Visual Studio.

DCLJobView A Windows Forms user control implementing all DCL frontend functionality.

DCLAddinIconLibrary This library is used to create a DLL which contains the toolbar icons. This is required for Microsoft Office Toolbars. The DLL is installed with the add-in installer. See the files CUSTOM_BITMAP_HOW-TO.TXT and README.TXT inside this project for detailed information.

DCLAddinInstallerClassLibrary This is an installer class which is used by the DCLAddinSetup on uninstalling. It is responsible for removing commands and especially the toolbar of the DCLAddin.

DCLController A small standalone application to show the usage of the JobViewControl in a different environment.

DCLAddinSetup The setup project for the add-in. Contains critical settings for the registration of the add-in.

DCLControllerSetup The setup project for the standalone application DCLController.

Rebuilding the DCLAddinSetup project creates a new up-to-date installer. All required projects are automatically built before.

For debugging the add-in it is possible to run it in a second Visual Studio IDE while the first one acts as the debugger. To enable this you first have to modify the path to the ”devenv.exe” on your system. Open the properties of the DCLAddin project, go to the configuration properties and change the application start command line for debugging. This path is typically ”VS_NET_INSTALL_DIR\Common7\IDE\devenv.exe”.

To setup the registry for the add-in the first time, build the DCLAddinSetup and run the installer. It is not necessary to re-run the installer if the DCLAddin sources are modified (because after building the add-in the new assembly is registered as COM object overriding the link to the installed assembly). Unfortunately this is not the case with the DCLJobView. It is advised to test modifications of the control in the standalone application because rebuild and startup time is notably shorter.

References

[1] Operating Systems & Middleware Chair. http://www.dcl.hpi.uni-potsdam.de.

[2] DCL. Distributed control lab. http://was.discourse.de.

[3] DISCOURSE. Distributed & collaborative university research & study environment. http://www.discourse.de.

[4] Gamma et al. *Design Pattern*. Addison-Wesley, 1996.

[5] Hasso Plattner Institute for Software Systems Engineering GmbH. http://hpi.uni-potsdam.de.