UNIVERSITY OF OSLO
Department of Informatics

Integrating Conduit with Windows Installer

Master thesis

Erik Solhaug Fløisbonn

23. april 2009

Contents

Chapter 1

Introduction

Conduit is a multi-platform distribution and deployment system originally created by Arve Knudsen. Conduit sets out to streamline the installation process of software, and assist in the installation of software that might otherwise be quite complex to install manually. The system is written in the Python programming language and consists of a generic design which is implemented on multiple operating through several Python modules.

The Conduit software suit consists of Python programs that handle the creation, distribution and installation of software packages. These programs provide facilities for the software provider to distribute their software, and for the user to deploy the software. Together their purpose is to offer the user an seamless operation of deployment.

1.1 Compatibility

The Conduit system was initially implemented on the Linux operating system, but is designed and now implemented, to support multiple operating systems. As a result of having compatibility for multiple operating systems, the system supports the most common features present in all operating systems, the common denominator, and neglects unique features present in some of the operating systems. The initial focus on Linux in development of the Conduit system is apparent in the features it supports. It is stated in the original documentation of the system that:

> "The focus on Linux during development was a conscious decision, a way to keep complexity down."[1, Introduction]

As a result of this, the system shows a skewness towards compatibility with Linux, in essence neglecting the unique features of other operating systems.

One such operating system is Windows, who differ from Linux in many regards. But with our focus on distribution and deployment of packages, the most clear difference of the two operating systems is the installation service shipped with Windows. This service is used by most programs being installed on Windows. The service provides the user with a familiar user interface when installing software, and a familiar location in Windows to remove software. It gives the software provider a standard way of packaging software, making the process easier than for operating systems like Linux, which has no standard installation service. And it makes the installation of packages easy for the users, as the packages are self-executable, and supported, out-of-the-box, by all modern versions of Windows[1].

1.2 Problem

To keep things simple and to hold support for multiple operating systems, the Conduit system uses its own installation and uninstallation routines. These are coded in Python, and are created and tweaked for each platform. An advanced feature like roll-back of installations is not present in Conduit's own deployment routines, meaning failed installations will not be removed correctly. This together with the problem of keeping the routines up-to-date as new versions of operating systems arrive, calls for the use of specialized software that is better suited to function properly. By utilizing the already present installation service in Windows, Conduit can use the service's advanced features, like roll-back, to become a more advanced and dependable system.

1.3 Specialization

The Unix Philosophy

"Make each program do one thing well" - Mike Gancarz [2]

The Conduit system has been built on the premise that it would be a generalized software concentrating on the concepts of distribution and deployment. As with most multi-platform software, the process of staying compliant may hinder the development of advanced specialized features. This does not however mean that the system, from the standpoint of the user, can not support advanced specialized features. The system can get support for such features through utilization of specialized software.

[1]Windows Installer is shipped with Windows Vista and Windows XP, and is installed by default. Older versions of Windows can install Windows Installer through service packs.

Most package systems has a distinction between software handling distribution, and software handling deployment. These systems are usually a collection of specialized software each dealing with either distribution of packages, or deployment of packages. One example of such system is Debian's APT.

By looking at another widely used package system we can find similarities and differences with the Conduit package system, and perhaps apply these differences to Conduit for the goal of becoming a more advanced software system.

1.3.1 The Advanced Packaging Tool

The Advanced Packaging Tool (APT) is a collection of tools that together form the system's functionality of distributing and installing software packages. APT functions as a front-end to dpkg, which is a program that solely handles the installation and uninstallation of .deb packages. Any dependencies that the software may have with other packages are not handled by dpkg, but by APT. This shows a clear distinction between the software handling distribution, and software handling installation. As the installation software is separated from the distribution software, other software similar to APT can have their own database of dependencies, and install .deb packages through dpkg. APT is a popular package system that is used by Debian Linux, a distribution (distro) of Linux, that other popular Linux distros are based on. APT has in recent years become quite popular as Debian based distros such as Ubuntu has adventured into the mainstream.

1.4 Conduitism

The tradition in development of software in Unix-like operating systems is to keep software simple, and to make software specialized. With a set of specialized software, an advanced system such as a package system can utilize the specialized software, and give the user an experience of a complete system. This can be done through having a system acting like a conduit between specialized software, each being relatively simple and doing their thing well.

Looking at Conduit the program that is most similar to APT, in functionality, is cdtdeploy[2]. It downloads the packages and any of its dependencies, and installs them on the user's computer. The difference is however that

[2]APT is a command line program, so visually cdtdeploy is more similar to front-ends to APT such Synaptic or Aptitude.

APT uses an external program for installation of each package, while Conduit also takes care of the installation.

By making cdtdeploy utilize the Windows Installer service to install the wanted packages, the deployment software will work as a conduit between the distribution software in the Conduit system, and the installation software on the user's machine. In essence following the tradition of separation and specialization.

Contrary to Conduit, APT is only functioning on Unix-like systems, but with its use of an external installation software it is clear to see that utilizing specialized software is not just normal, but the way for generalized systems to support specialized features.

1.5 Windows Installer

Windows Installer is the native installation and configuration service for the Windows operating system. It is installed by default on Windows XP and Vista, and is available for installation through service packs for earlier versions of Windows. Windows Installer uses packages with file extension .msi, which more specifically is a relational database containing information about the installable files together with the installable files themselves. The installation service provides support for platform specific operations like editing of the windows registry and creation of shortcuts, and supports features such as uninstallation and rollback of a failed installation.[3]

Windows Installer runs as an active service in Windows, and because it is shipped with most modern Windows versions, we can assume that is it running at all times. The fact that it is shipped with Windows XP/Vista, and running by default simplifies our situation greatly compared to situations where you are utilizing external software that may not be present. You usually resolve such problems by bundling the software, or requiring the installation of the software beforehand. But this is only necessary with Windows versions older than Xp, in which case Windows Installer is available through service packs.

1.6 Integration

"To make into a whole by bringing all parts together; unify." – Definition of integrate by The American Heritage[4].

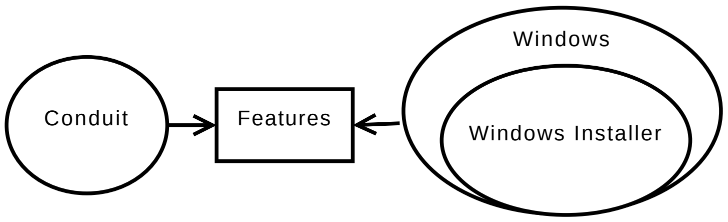


Figure 1.1: The integration of Conduit with Windows installer. The feature entity represents the features that the integration will produce for the user.

By seeing that the Conduit system can get support for new advanced features, by adding support for specialized software such as Windows Installer, we can look at the process of integration explained in the rest of this thesis. With integration we mean to bring features currently available in Windows Installer to the Conduit system, and to bring features of Conduit to the Windows environment through the use of Windows Installer. We start off with Conduit and Windows Installer as two separate parts, they are not utilizing each others features and do not know of each other.

In this thesis we will present what we believe is the most suitable integration of Windows Installer and Conduit pertaining to the distribution and deployment of software packages. We will show what parts of Windows Installer that Conduit should utilize, and what parts of Conduit Windows Installer should utilize. This will result in piece of software, and will be presented as a new version of Conduit containing the changes discussed in this thesis.

1.7 Guiding points

"Make everything as simple as possible, but not simpler." – Albert Einstein

In my pursuit for integration of Windows Installer I have set some guiding points which I have followed religiously throughout the development process. These are:

Keep compatibility

Because Conduit is a multi-platform software system, the integration of Windows Installer should not diminish or destroy compatibility with other platforms. It should instead increase the compatibility with the Windows platform and strengthen the reliability of the system.

It would be very tempting to redesign certain parts of the Conduit system to better suit the utilization of Windows Installer, but as the other operating systems do not have a similar installation service, it is better to use the current design to keep compatibility.

Keep things simple

As Conduit, and Windows Installer particularly, are big software systems, they offer many features, both simple and complex. And the utilization of these can features can therefor span from shallow to full-blown. With the introduction of Windows Installer to Conduit, I have looked at the current Conduit system and tried to utilize Windows Installer the same way that Conduit operates internally. This has kept me from introducing a lot of new concepts to the Conduit system, and has allowed me to keep things simple and general.

A good tangible example of how I have kept things simple is the connection between the Conduit database of installed packages, and Windows Installer's database of installed software. Windows Installer runs as a service, and has it's own database with the listing of installed software installed through Windows Installer. This list could be a complement to the database Conduit is maintaining with installed software. But as the relationship between the two databases would become quite complex, it is better to concentrate on the distribution and deployment phase with creating packages and installing them, and ignoring, at this point, the problem of finding software already installed on the computer through Windows Installer.

Divide and conquer

By dividing a problem into smaller subproblems you can concentrate on smaller specific problems without being flabbergasted by the overall problem. In our case this applies to our approach to the Conduit system, where we divide it into categories, and for each category tries to find the optimal solution pertaining to the integration of Windows Installer.

1.8 Thesis structure

In this thesis I will follow a top-down approach. I will start by stating my vision for a Conduit system with the wanted degree of integration with Windows Installer. This will be followed by a chapter explaining the current design of the Conduit system together with my changes. The chapter is followed by a chapter explaining the specifics of the changes, introducing the complex nature of Windows Installer and explaining my approach to the integration programmatically. The thesis is then concluded with a look at my results, ending with a description of possible future improvements to the Conduit system.

Chapter 2

Vision

The Windows Installer software offers many services that can be utilized in the Conduit system. With this in mind, I present a vision that the integration of Conduit with Windows Installer will focus towards. My vision is that:

- The Conduit system will utilize the services provided by Microsoft Installer to become a better and more reliable deployment system.

To keep my vision concerning my integration with Windows Installer reachable and realistic, I have four overall goals for the integration:

1. Get the Conduit system to utilize the Windows Installer service to better the process of deployment by adding support for advanced features only present in state-of-the-art software.

2. Get better support for platform specific operations, with emphasis on keeping the focus on the Conduit system to general distribution and deployment of software.

3. Get a seamless integration with the Windows Installer software seen from the standpoint of the user.

4. Use Windows Installer packages interchangeably with the current cdt-package packages.

If the the goals above are met, to a degree that coincides with the guiding points presented in the previous chapter, we feel that the Conduit system will become a better and more reliable deployment system compared to the current system.

2.1 Advanced features

The Windows Installer service is a specialized software that is dealing with packaging and installation of software. In these processes the service offers advanced features that is hard to implement for smaller software. These features include a GUI framework, automatic generation of uninstallation scripts (roll-back) and advertisement of installation with installation on demand.

Not all of these features are relevant to the Conduit system. The GUI framework enables the package creator to specify a user interface that shows up when installing packages. This is not needed by Conduit as it has its own user interface, which should not be interfered by other windows popping up. Advertisement of installations will also not be utilized by the Conduit system as it is a way for a package to install only certain bits of a package and make the rest be installed on-demand.[**?**]

Automatic generation of uninstallation scripts (rollback) is an advanced feature that Conduit can utilize. Windows Installer creates a undo-script adding a undo-action for each action done in an installation. Together with this script Windows Installer saves a copy of each file that is deleted during installation. When an installation fails, Windows Installer performs a rollback installation that returns the system to the state it was in before installation.[5]

All the normal actions specified in a Windows Installer package has an corresponding undo-action, which means that a rollback-script is generated automatically. By utilizing the Windows Installer service in Conduit, the system will get the rollback feature without having to worry about the actual process of generating a rollback script.

2.2 Platform specific operations

If we look at the three supported operating systems supported by the Conduit system, Windows, Linux and Mac OS X, they all operate with much of the same concepts such as environment variables and shortcuts. They do however differ in their implementation of these concepts, and it makes it hard for generalized software to use these concepts uniformly. To make things even worse, without having a standard way of using these concepts the use of specialized code might be hard to maintain as implementations change. To avoid this situation, a good solution is for the generalized system to use a specialized software to handle the platform specific operations.

With the introduction of Windows Installer packages the Conduit system can describe the installation of a software and let the Windows Installer service handle the details of the actual installation. This is done by letting Conduit describe the structure of an installation through a Windows Installer package. With this approach Conduit can focus on what to be installed and let Windows Installer focus on how it should be installed. As Windows Installer is created and constantly maintained by the creators of the operating system, this approach assures that the installation of platform specific operations are done properly.

2.3 Seamless integration

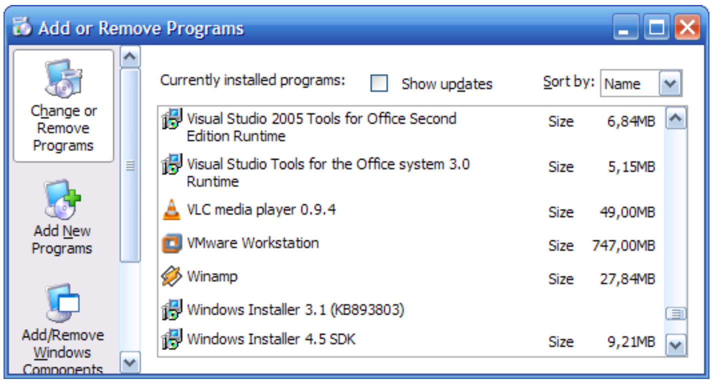
The ordinary user of the Conduit system sees the utilization of Windows Installer mostly from a user interface perspective. As stated above the design of the user interface will not be utilizing the GUI framework offered by Windows Installer, but instead stay consistent with other platforms. As a result, the changes done to the user interface to fit the incorporation of Windows Installer software can be an indication of how seamless the integration has been for the user. With this in mind we can state some goals of how we want to present the integration for the user:

1. Progress of installation

   The current Conduit system guides the user through the installation of software. This is done through a user interface that shows the progress of an installation as each package is installed. This progress is updated through a callback function that the installation software uses to update the progress bar. By using an external software this process might not be possible in all cases, as the progress needs to be conveyed between two separate programs when one of them are installing the package. This is a complex task, and without a Windows Installer API offering external UI the task would be very difficult. But as the Windows Installer offers an API that is capable of this, utilizing the interface correctly will result in showing the progress of installation the same way as with cdtpackage packages.

2. Install with Windows Installer

   With proper integration of Windows Installer, the Windows Installer packages installed through the Conduit system should be listed in the location where Windows lists its installed software. This location is embedded in the Windows operating system, and is called add/remove software (see figure 2.1). The Conduit user should be able to remove the packages installed with Conduit in this location. By doing this we are integrating the Conduit system into the Windows operating system and allowing the users to remove software in one familiar location.



Currently installed programs: ☐ Show updates Sort by: Name

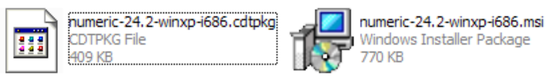
Visual Studio 2005 Tools for Office Second Edition Runtime	Size	6,84MB
Visual Studio Tools for the Office system 3.0 Runtime	Size	5,15MB
VLC media player 0.9.4	Size	49,00MB
VMware Workstation	Size	747,00MB
Winamp	Size	27,84MB
Windows Installer 3.1 (KB893803)		
Windows Installer 4.5 SDK	Size	9,21MB

Figure 2.1: The standard location for Windows users to remove installed software.

2.4 Microsoft package and cdtpackage

To utilize the Windows Installer service in the Conduit service we need to add support for the Windows Installer package format (.msi). This package format is capable of holding the same data as the cdtpackage format, as it consists of a relational database and an archive (.cab) holding the actual files. The relational database is able to store custom tables that are not used by Windows Installer, which makes us able to store metadata and the same, or more, of the data that is currently in a cdtpackage package.

With this in mind it is quite reasonable to say that we should be able to use the current cdtpackage and Windows Installer packages interchangeably in the Conduit system. This involves adapting a similar interface to that of cdtpackage to the Windows Installer packages, and letting the Conduit system handle the installation of a package in a uniform way.



numeric-24.2-winxp-i686.cdtpkg
CDTPKG File
409 KB

numeric-24.2-winxp-i686.msi
Windows Installer Package
770 KB

Figure 2.2: The Conduit cdtpackage package and the corresponding Windows Installer package.

2.5 Realizing the vision

With the goals of support for advanced features, platform specific operations and seamless integration to the user in mind, we can look at what changes has to be made to the system to realize the presented vision. In the next chapter the design of the Conduit system will be revised, with the purpose of creating the structure needed to move towards the vision explained in this chapter.

Chapter 3

Design

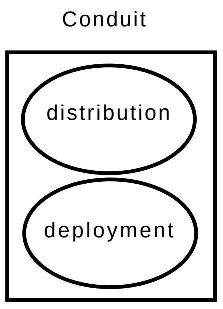


Figure 3.1: The Conduit system with abstract categories.

The Conduit system consists of a collection of programs which mostly fit into two categories: distribution and deployment. The programs that present facilities for the software provider to publish their content through the Conduit system, are put in the distribution category. The programs that enable the user to download and install the software added by the software providers are put in the deployment category.

The reason for compartmentalizing the Conduit system into categories is simple: it will enable us to look at parts of the Conduit system and concentrate on only the parts that are relevant to us. By doing this we are in essence narrowing down our area of interest, and as a result, giving ourself a better chance at grasping a complicated software system.

3.1 Distribution

The Conduit system has several programs dealing with distribution. These include the web portal where the software provider controls his software, and

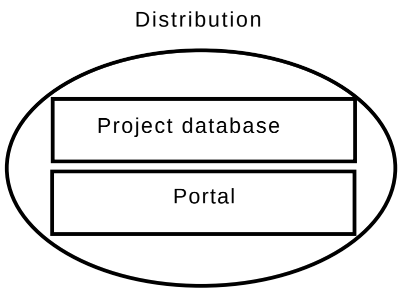


Figure 3.2: The distribution category with software facilitating distribution for software providers.

the project database where the description of each project is stored. All of these programs facilitate distribution of software, and they are mainly used by the software providers or Conduit maintainers. However the service that the project database provides is used by the deployment software, which means that the distribution software acts like a service for the deployment software.

3.2 Deployment

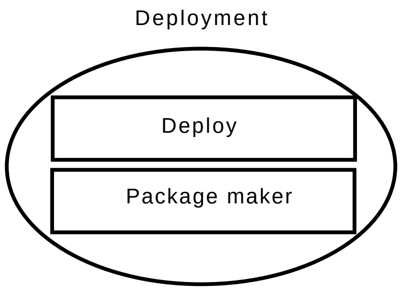


Figure 3.3: The deployment category with software facilitating deployment of software for the user.

The software in the Conduit system that gives the user the ability to download and install software given by the software providers, is in the deployment category. These programs include cdtdeploy and cdtpkgmaker. The cdtdeploy program is responsible for the actual deployment of packages, from downloading of packages to installation and uninstallation. Cdtpkgmaker is the program responsible for the creation of software packages. Even though this program is not used by the Conduit user, but by the software providers, it is placed in the deployment category as it uses the facilities offered by the software in the distribution category.

3.3 Concentration

With the Conduit system divided into distribution and deployment it is quite easy to see that the process of integrating Windows Installer with Conduit must mainly be focused towards the software in the deployment category. In fact, with the guiding points and goals explained in previous chapters in mind, the integration of Windows Installer with Conduit will not include the distribution software.

Windows Installer is an installation service, and you might say that it is obvious for the integration of this to apply only to the deployment software. But with the addition of the Windows Installer packages, it may not be so obvious, as they bring their own concepts not found in the Conduit cdtpackage format. But by creating relations between the package formats, and stating that Windows Installer packages must adhere to some specific rules, we can use the packages interchangeably and we can avoid making changes to the distribution software.

By not changing the distribution software we can be sure that the distribution part of Conduit will stay compatible with other operating systems. And more importantly, it will force the integration of Windows Installer to follow the design of the distribution software. This will avoid infesting the distribution software with platform specific concepts.

3.4 Deployment software

In the process of incorporating Windows Installer into the Conduit system there are two main processes to look at: the creation of the Windows Installer packages, and the installation of the packages. To be able to add support for the creation and installation of such packages, their corresponding software need to be looked at in detail. To what extent changes is needed is only apparent after looking at the design of each program. We will start with cdtpkgmaker.

3.4.1 cdtpkgmaker

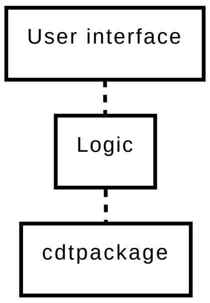


Figure 3.4: The design of cdtpkgmaker, with user interface (ui), logic and Conduit package format (cdtpackage).

The cdtpkgmaker program consists of a user interface (ui) guiding the user through the process of creating a software package for the Conduit system. This ui cooperates with a logic to enable the program to create a software package. It does this by letting the ui have several defined stages. The most relevant stages are:

1. Entering of package info

   This stage lets the user specify values for the software release such as project file, version of release and the address of the source for the software package.

2. Creation of package

   This stage has no user input and uses the cdtpackage software to create a Conduit package. As the program is hardcoded to use the cdtpackage package format, it uses this software to create a package ready to be uploaded.

   (a) Download source package

      This step downloads the source package described in the project file to the computer. The source is used by the Conduit build system to build a software package.

   (b) Build software

      This step looks at the build method for the software, and builds the software. This is done either by a configure & make process, or through Python distutils with setup.py. When the building of the software is complete, the software is ready to be packaged.

   (c) Packaging

This step packages the built software with the Conduit cdtpackage format. This step is hardcoded to use the cdtpackage format, and results in a .cdtpkg file that is ready to be uploaded.

3. Upload package to database

   This stage finds the path of the finished package and uploads it to the package database, together with updating the project description of the release so that it points to the newly created package.

The most interesting stage, which we can change to accommodate for the support for Windows Installer packages, is stage 2, and more specifically step b. Stage 1 and 3 are not dependent on the fact that the created package is cdtpackage, as long as the windows installer package is able to store the same data and metadata.

As Conduit is a multi-platform system, we must add support for the Windows Installer package while still supporting the creation of packages on the cdtpackage format. We need to find a way to decide what package format to use on-the-go, with constraints to what operating system the user is using, and what package format he wants to use. There is however no reason in making this process complicated, as the easiest and best solution is for the user to decide what package format he wants. If the user is on a non-compliant operating system, such as the scenario where he wants a Windows package while using Linux, the system simply creates a cdtpackage with the .cdtpkg extension.

To get cdtpkgmaker to work with the Windows Installer service the cdtpkgmaker software has to change its design slightly to also include Windows Installer package software. The ui does not need to be changed - as explained above: the process of choosing the correct package format is done automatically. The logic however must be changed to also work with the Windows Installer service. But as the cdtpackage software is originally separated from the internal logic of cdtpkgmaker (see figure 3.4), we do not need to extract the cdtpackage software from the logic. But can instead create a similar type of software for the Windows Installer package format, and let the program choose among them at runtime.

The new design (Figure 3.5) of cdtpkgmaker has an extra entity representing the software responsible for the creation of Windows Installer packages. With the new design ready we are able to concentrate our efforts towards the process of developing the Windows Installer package software programmatically.

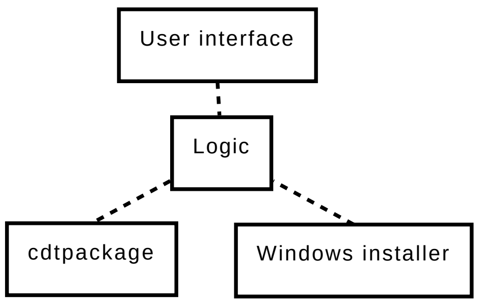


Figure 3.5: The revised design of cdtpkgmaker, with user interface (ui), logic, Conduit package format (cdtpackage) and Windows Installer package format.

3.4.2 cdtdeploy

Cdtdeploy is the program that the ordinary user of Conduit uses to install software from the Conduit system. This program lists, installs and uninstalls software from the Conduit project database. As with the cdtpkgmaker program, this also operates with defined stages which guides the user through the process. These stages are:

1. Listing of packages

   This stage lists the packages available to be installed on the user's computer, taking into account the operating system and computer architecture.

2. Installation of packages

   This stage acts on the input given in the previous stage. With the releases selected, the program goes through the process of installing each release in a matter which resolves dependencies. The installation of a package has 4 steps. These are:

   (a) Installation of files: This step copy files from the package to the correct directory on the user's computer.

   (b) Update OS environment: This stage updates the operating system environment to work correctly with the operation of the Conduit system.

   (c) Update Python path: This stage updates the Python path so it points to any new installed Python modules.

   (d) Start menu: This stage updates the start menu by adding items described in the package

3. View of installation

> This stage shows the files copied through the installation stage together with a description of the changes done to the environment.

As this program works as the front-end to the user for deployment of software in Conduit, it is important that the program stays user friendly, and consistent, regardless of what platform and package format is being used. As a result, the user interface will not be changed - it will keep the same stages. However the four steps of stage 2 might be fused together as these steps will be handled by Windows Installer and not Conduit directly.

The old and new design of cdtdeploy is similar to that of cdtpkgmaker described above (Figure 3.4, and Figure 3.5). This fact gives us the possibility to develop the internal workings of the Windows Installer software independently from the logic of the Conduit software. We can do this by adapting the same API as that of the cdtpackage software, to the Windows Installer software, and let the deployment software use the package formats interchangeably.

Chapter 4

Specifics

With the revised design of both deployment programs in the Conduit system ready, we can go into more specific details surrounding the process of integrating Conduit with Windows Installer. As explained earlier my vision is for the cdtpackage format to be used interchangeably with the Windows Installer packages. For this to become a reality we need to get familiar with the concepts of Windows Installer, and try to set relationships between these and the concepts of the Conduit system.

4.1 Windows Installer

The Windows Installer service divides installable software into four entities: package, product, feature and component (see figure 4.1). A Windows Installer package describes the installation of products, and is manifested in the form of a distributable file. These packages use the file extension .msi and consist usually of a Cabinet archive (.cab) together with a relational database holding installation data pertaining to the files in the archive.[6, Installation Package]

The product described in a package is a set of programs, and is the most general entity for Windows installer which can be uniquely identified, ignoring the package. A product is composed of components, which are the

Windows Installer:	Package	Conduit:	Package
	Product		Release
	Feature		Interface
	Component		

Table 4.1: Terms used by Windows Installer and the Conduit system.

atomic entity for what Windows Installer is concerned. Components are the data, such as files and registry keys, that a program is made out of. The feature entity is a collection of components and features, and is usually one separate part of the total functionality of a program.[6, Components and Features]. Figure 4.1 shows graphically the relationship between the terms used by Windows Installer.

Conduit and Windows Installer both operate with packages as software converted to a certain format ready for deployment. In Conduit a package is a realization of a project release, and it has no concept of entities smaller than a release. In comparison, a package in Windows Installer is a set of not-necessarily-related products, which depends on smaller entities such as components. As of such, confusion might ensue as to what a release in the Conduit system corresponds to in a Windows Installer package.

Since a Windows Installer package may contain more than one product, a release in Conduit is best compared to a product in Windows Installer. This however leads to problems with the fact that a Windows Installer package can contain several products. These products may, in theory, not be related, and this may lead to problems, as a package in Conduit can only be said to be a release of a certain project. There are two ways to resolve this issue. Either we inject the project database with platform specific data and allow a Windows Installer package to have multiple products, and in essence change the current design of the distribution software. Or we can simply say that a Windows Installer package, created with Conduit, can only consist of one product. This way a product will correspond correctly with the dependencies specified in the Conduit package database, and with the Conduit's general concept of a release.

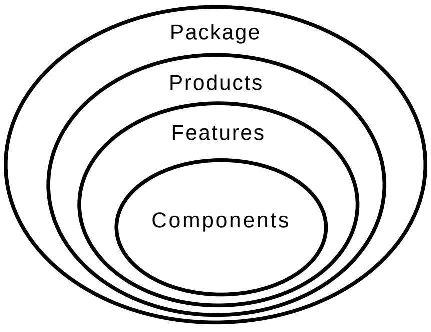


Figure 4.1: Showing the relationship between terms used with Windows Installer.

By saying that a single product in Windows Installer corresponds to a release in the Conduit system, we get a consistent definition of the term release. As a result, the Conduit program can work with both package formats the same

Table	Description
Component	Lists installation components
Directory	Directory layout for the application
Environment	Lists the environment variables
Feature	Defines the logical tree structure of features
FeatureComponents	Defines features and component relationships
File	Complete list of source files with their attributes
Shortcut	Lists information needed to create shortcuts

Table 4.2: Tables used by Windows Installer to describe installable data. Excerpts from [7, Database Tables]

way without introducing disparities. By having one product per Windows Installer package, the dependencies stated in the Conduit-release description will be valid for Windows Installer packages, and can consequently use Conduit's dependency resolver. Furthermore the two package types become interchangeable, and the only difference in deployment is that they invoke different package software systems depending on the package format.

To summarize, Windows Installer packages created with Conduit will contain the following entities:

- Only 1 product. This product corresponds to the release in the Conduit system, and to the software in a cdtpackage package.
- Only 1 feature. This feature is the default one, which contains a reference to the component of the root directory.
- Components. These components reference directories and files of the software.

4.2 Creation and installation

Now that we have specified the relationship between the package formats, the next step is to go into more details of how the actual packaging and installation of Windows Installer packages will function. To explain this, we need to get an understanding of what the Windows Installer packages consist of. As explained before, a Windows Installer package consists of a relational database. This database consists of several tables that are used in describing the installation of a package. Besides having tables pertaining to the actual installable data, it also has tables explaining the sequence of installation, and description of the user interface. As Windows Installer contains a myriad of tables, this section will focus on the tables describing the data being installed. The remaining tables will not be described as they

Column	Type	Key	Nullable
Component	Identifier	Yes	No
ComponentId	GUID	No	Yes
Directory_	Identifier	No	No
Attributes	Integer	No	No
Condition	Condition	No	Yes
KeyPath	Identifier	No	Yes

Table 4.3: Component table describing a component in the Windows Installer package. Taken from [8, Component Table]

Column	Type	Key	Nullable
File	Identifier	Yes	No
Component_	Identifier	No	No
FileName	Text	No	No
FileSize	DoubleInteger	No	No
Version	Version	No	Yes
Language	Language	No	Yes
Attributes	Integer	No	Yes
Sequence	Integer	No	No

Table 4.4: File table describing a file in the Windows Installer package. Taken from [9, File Table]

will not be altered by us directly, but will be used with their default values. This is mainly because we will use the Conduit system's own user interface while installing the package, and there is no practical reason for changing the order of installation, or adding custom actions. The tables pertaining to the data are listed in Table 4.2.

The Component table holds information about the components in a Windows Installer package. It has the columns described in table 4.3. The Component column is the key column for the table, and it uniquely identifies a component. The File table holds information about files, and it has the columns described in table 4.4.

Without going into too much details about every table, it's columns and their corresponding types, the files in a Windows Installer package are related to a component through the Component_-value in the File table. This happens through a primary/foreign key fashion, and this is how all the tables relate to each other in a Windows Installer database. The feature/component relationship is stated as a row in the FeatureComponent table refering to the component and feature. Directories, environment variables and shortcuts are represented as rows in their respective tables.

4.2.1 Get all files in an installation

To get a better grasp of the structure of a Windows Installer database, I will present an algorithm for the listing of all the files in an Windows Installer database.

Algorithm: Given a Windows Installer package, get a list of all the files referenced in the relational database.

1. Get root directory: A root directory is a row in the Directory table which has no Directory_parent, or has Directory_Parent set to itself. Add directories to list *d_list*. Go to step 2 with *d_list_tmp* ← *d_list*.
2. For each item *i* in *d_list_tmp*: Get a list *d_tmp* with all directories in table Directory with Directory_parent equal to *i*. Append *d_tmp* to *d_list*. If *d_tmp* is empty: go to step 3. If not: go to step 2 with *d_list* ← *d_tmp*.
3. For each item *i* in *d_list*: Add all components in Component directory which has Directory_ set to *i* to list *c_list*.
4. For each item *c* in *c_list*: Get all files in the File table that has Component_ set to the *c*.

One ends up with a list of all the files that is referenced directly, and indirectly, from the root directory. If one wants the path for each file, one can add properties to the individual objects, and append the path as one is recursively adding directories in step 2.

4.2.2 Tools

From a practical standpoint, splitting a tangible software into a chaos of rows in a database, is not the optimal situation for small software. The complexity of Windows Installer makes the process of creating a simple installer package quite difficult. Without the help from tools, the task would be mind-boggling. There exists however several commercial and a few gratis tools that can be utilized for the creation of a package. A free toolset created by Microsoft called WiX[10] exist, as well as a more commercial option such as Installshield[11]. The Conduit system is closely coupled with the Python programming language, and thankfully Python 2.5 and onwards comes bundled with a low-level Windows Installer library called msilib. As the msilib module is featured in the Python standard library, msilib is the obvious choice for the Conduit system. By using msilib we avoid adding dependencies to external programs besides Windows Installer, which we already have assumed is present and running on the user's computer.

4.2.2.1 Msilib

Msilib is a Windows specific Python module that supports the creation of .msi files[12]. The library consists of a set of routines that can be used to create and read windows packages. The msilib library routines are quite low-level, meaning the correspondence between the Python methods and the wrapped Windows Installer functions are high. This enables the package creator to get full control of the package process. It does however mean that the package creation process can get quite complicated. With the added bonus of practically no documentation of the usage of the msilib module, the development process of the packaging software to this thesis has been lead by a period of trial-and-error. The knowledge gained from this process is presented in the following sections of the thesis and in the Conduit system, and can hopefully function as a reference for future users of the library.

4.3 Package creation with msilib

According to the documentation of msilib, in the Python standard library, "the package contents can be roughly split into four parts: low-level CAB routines, low-level MSI routines, higher-level MSI routines, and standard table structures."[12]. In my package creation software using msilib, I have used the low-level MSI routines, higher-level MSI routines and the standard table structures. The low-level CAB routines are used indirectly through the use of higher-level MSI routines. The process of creating a Windows Installer package compatible with the Conduit system has been split into three steps:

- Initializing package
- adding installable data
- finishing package

4.3.1 Initializing

The first step is to create a Windows Installer database that we can work with in the following steps of the process. This is done by calling the msilib method init_database(), which initializes the database with a given schema. It adds product name, product code and product version to the package. The schema used is the one from the msilib module, which holds the definition of the tables needed for a working Windows Installer database. The initialization step is ended by adding table contents to the various sequence tables - the tables responsible for the sequence of installation.

4.3.2 Adding installable data

The next step is to add installable data to the Windows Installer package. This process has been split into the following steps:

1. Default feature

   This step adds a default feature to the Windows Installer database. This feature is be the only feature added to the Windows Installer database and it references all the components holding installable files.

2. Cab file

   This step adds a .cab file to the database which holds the actual content of the installable files. This cab-file is referenced by the components to show what data the component consist of.

3. Root directory

   This step adds a root directory to the database. The source directory is set to SourceDir and the target directory is set to TARGETDIR. By doing this we are specifying that this directory is the root directory for the installation. When the Conduit system is installing the package, the TARGETDIR property is set by Conduit to the root of the installation path.

4. Files

   This step adds all the files in the directory that is to be packaged. We do this by adding a directory and a corresponding component to each directory we meet while traversing the path. This has to be done top-down, beginning with the root directory, as all the directories in the root directory reference their parent directory in the Directory_Parent column of the Directory table. When we encounter files, they are added to the File table with a reference to the component that belongs to the current directory.

5. Shortcuts and environment

   As the files are added to the database, we can now add external references to these files in the form of shortcuts and environment variables. If the software has a recipe saying it should have a shortcut, a reference to the component of the file is added to the Shortcut table. Any directory with library files, Python modules and executables are added to the Environment table. This table allows us to append text to environment variables, such as PATH, enabling the user to run executables without knowing their full path.

Category
User interface and logging
Handle management
Installation and configuration
Component-specific
Application-only
System status
Product query
Patching
File query
Transaction management
Database functions

Table 4.5: The categories that the Windows Installer functions can be divided into.[13]

4.3.3 Finalizing

We now have a Windows Installer database with information pertaining to installable data and sequence of installation, together with a .cab file containing the actual installable files. Together they form a Windows Installer package we want to add to the Conduit system. To finalize the package we commit the changes to the cab file, together with the changes made to the database. This results in a .msi Windows Installer package which is ready to be uploaded, and be used in the description of a project.

4.4 Package installation

With a Windows Installer package ready we will now look at the process of installing a .msi package. Like the situation with the creation of windows installer packages described above, there are several ways to go in installing a Windows Installer package. Users of a Windows operating system might be familiar with the method of double-clicking a .msi file, and being shown a wizard-like user interface. When such a double-click occurs, the msiexec application bundled with Windows is invoked with the default options of an installation. The msiexec application uses the Windows Installer application programming interface (API), that the Windows operating system offers, to show installation UI and start the installation. To get a similar process in Conduit we need to get familiar with the Windows Installer API.

4.4.1 Windows Installer API

The Windows Installer API consists of a set of installer functions that one can use to add Windows Installer functionality to an application. The installer functions can be divided into the categories, listed in table 4.5, as they are done in the installer function reference by Microsoft[13]. As we want to install a package, and follow the progress, the most interesting functions are the ones belonging to installation and configuration, and user interface and logging. By using the functions in the installation and configuration category, we can initiate an installation of a package. And with the user interface and logging functions, we can specify that we want to have our own user interface, and use callbacks to Python to keep track of the progress of installation.

The API is there with the capabilities of installing packages and giving feedback to Conduit on the progress of the installation. But now comes the question of how to utilize the API. We can use:

1. Automation objects

   We can work with COM and automation objects offered by Windows Installer by using the win32com[14] Python module.

2. Windows shared libraries

   We can use a foreign function interface (ffi), such as ctypes[15], and use the shared libraries that comes with Windows. disregarded

According to the Windows Installer automation interface reference[16], installer objects does not have the ability to use an external user interface. This is needed as the current Conduit system shows the progress of installation by specifying a callback function. As we want to leave the user interface alone, lessening the feedback to the user is not a solution and automation objects is thereby disregarded as a solution.

Using shared libraries directly with ctypes, as ffi, gives us the ability to specify an external user interface through the MsiSetExternal() method. Conduit can initiate an installation, say that it will not want the internal user interface, and specify a callback function that is called by Windows Installer as the installation progresses.

4.4.2 Installation with ctypes

The ctypes module allows us to use shared libraries in Python by introducing C types such as pointers. With these types we can define a callback method that has the same arguments and return value of that which is needed.

The process of installing a Windows Installer package with ctypes does not involve many calls to the installer functions. All we need is to initiate installation, tell it to use our own user interface and specify a callback method. This is done in the following way:

1. MsiSetInternalUI:

   This method is used to suppress the internal user interface. By specifying the constant INSTALLUIULEVEL_NONE, Windows Installer will not show any user interface other than the internal one.

2. MsiSetExternalUI:

   This method is called to specify our own callback method. The callback method is defined in Python, and through the use of ctypes, we create a class that calls our function on our behalf. This ui is used in both installation and uninstallation.

3. MsiInstallProduct:

   With the internal user interface suppressed, and with a callback function ready, we can initiate the installation of the package with this method.

By doing these calls we let Conduit initiate the installation of a Windows Installer package. With the callback we let the cdtdeploy program be notified that progress has been made in the installation. Features such as roll-back is initiated automatically when an installation has failed, or is canceled by the user. User interaction under installation is controlled by return values from the callback method. If the user wants to cancel the installation, the callback method checks the Conduit ui for such an action. If it is found, the callback method returns a cancellation value to Windows Installer telling it to stop the installation.

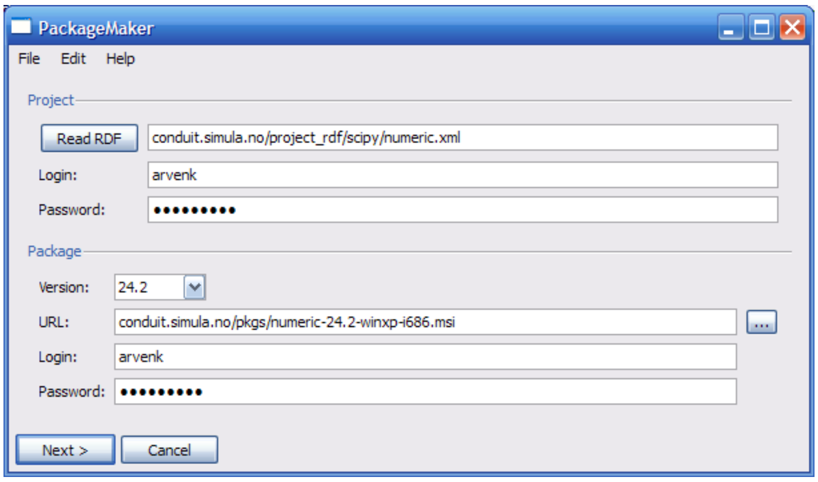
With the application of the methods described in this chapter to the Conduit system, we are adding support for Windows Installer features. This is done in such a matter that it allows Conduit to use different package formats, and packaging software, and still keep the user interface consistent between operating systems.

4.5 A working example

A working Conduit system, with the support for the Windows Installer explained in this thesis, is available at the following url:

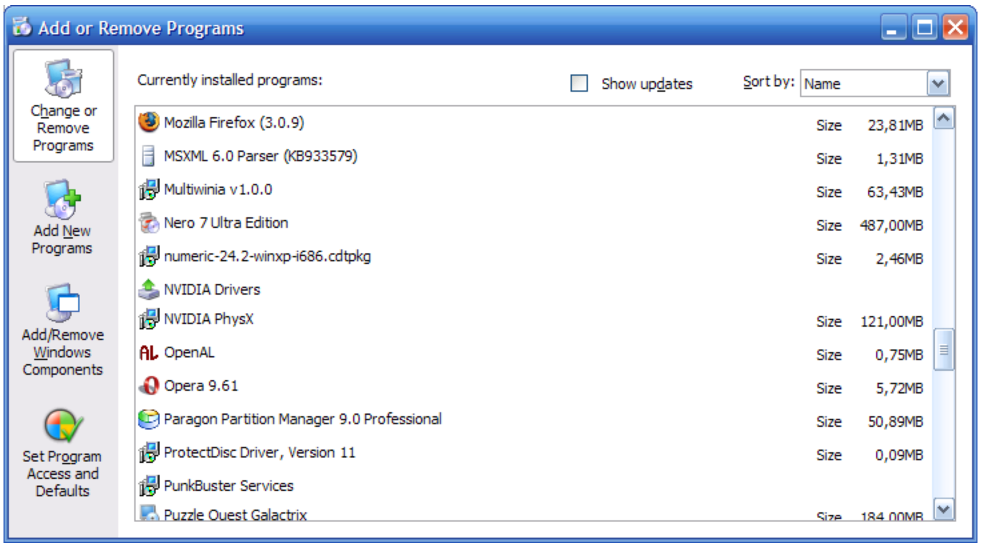
`http://heim.ifi.uio.no/erif/Conduit/`

There you will find the documentation for the Windows Installer package software, as well as screenshots showing examples of how Conduit is integrated with Windows Installer.



conduit.simula.no/project_rdf/scipy/numeric.xml
Login: arvenk
Password: •••••••••
Version: 24.2
URL: conduit.simula.no/pkgs/numeric-24.2-winxp-i686.msi
Login: arvenk
Password: •••••••••

Figure 4.2: The cdtpkgmaker application about to create a .msi package.


Add or Remove Programs
Change or Remove Programs
Add New Programs
Add/Remove Windows Components
Set Program Access and Defaults
Show updates

Currently installed programs:

Sort by: Name

Mozilla Firefox (3.0.9)	Size	23,81MB
MSXML 6.0 Parser (KB933579)	Size	1,31MB
Multiwinia v1.0.0	Size	63,43MB
Nero 7 Ultra Edition	Size	487,00MB
numeric-24.2-winxp-i686.cdtpkg	Size	2,46MB
NVIDIA Drivers		
NVIDIA PhysX	Size	121,00MB
OpenAL	Size	0,75MB
Opera 9.61	Size	5,72MB
Paragon Partition Manager 9.0 Professional	Size	50,89MB
ProtectDisc Driver, Version 11	Size	0,09MB
PunkBuster Services		
Puzzle Quest Galactrix	Size	184,00MB

Figure 4.3: The add/remove location showing the numeric package which has been installed through Conduit.

Chapter 5

Conclusion

5.1 Vision

In this chapter we will reflect on the new Conduit system resulting from the process described in the former chapters. This will be done by looking at our initial goals, and seeing to what degree they have been reached. This will give a measure of how well the integration of Conduit with Windows Installer has been.

With the utilization of the Windows Installer service described in this thesis, the service is used by Conduit to improve its methods of deployment. But as Windows Installer can offer other complementary features to that of Conduit, future development of Conduit can be focused around getting a fuller integration of Windows Installer features.

5.2 Evaluation

By looking at the new Conduit system we can see how well the integration process has realized our vision outlined in chapter 2. The vision is for the Conduit system to utilize the services provided by Windows Installer to become a better and more reliable deployment system. To see if we have realized the aforementioned vision, we can to see if we have reached the goals set in chapter 2.

5.3 Deployment process

The new Conduit system utilizes the Windows Installer service to deploy Windows Installer packages. This is done by utilizing the API that Windows Installer offers. By installing Windows Installer with the help of Windows Installer, the Conduit system is getting support for advance features such as roll-back. It is also getting better support for platform specific operations such as changing environment variables, by allowing Conduit to describe the data in a package, and let Windows Installer do the operation. With Windows Installer supporting an external user interface, the integration of Windows Installer is seamless from the standpoint of the user as it allows Conduit to keep its consistent user interface throughout operating systems. And by adapting the API used by the cdtpackage format to the new Windows Installer package format, we are allowing the Windows Installer packages to be used interchangeably with cdtpackage packages. The reliability of the Conduit system is improved by allowing for rollback of any unwanted changes done through a failed installation. In conclusion we feel that the goals outlined in chapter 2 has been met, and to such a degree that Conduit has become a better and a more reliable deployment system.

5.4 Future development

With reaching our goals of integration, the following section explains possible future improvements to the Conduit system pertaining to the integration with Windows Installer. Even with our goals of integration with Windows Installer met, it does not mean that the integration is total. There are some features that are not currently utilized, and some of the services currently utilized can be improved further.

5.4.1 Add/Remove software

With the utilization of Windows Installer to install windows packages, the Windows operating system notices what software that Conduit installs. This allows Windows to list the installed programs in a location called add/remove software, which is the standard place for users of Windows to remove software from their system. The new Conduit system has its installed packages listed in this location, and the user can remove packages installed through Conduit this way. The problem is however the connection between the Windows Installer database, and Conduit's database of installed packages. By removing software installed through the new Conduit, at the Add/remove location, the software is removed from the system but Conduit is unfortunately not notified of this. This basically means that Conduit thinks that the software is still installed, and lists it normally as installed, even though the software is not actually present. To get Conduit notified of the removal, Windows Installer can use a custom action to run an embedded JScript/VBscript, or run cdtdeploy directly with command line arguments describing the removed package. This approach however depends on us knowing where the Conduit system is located. Usually the user of Conduit only downloads the cdtdeploy application as a standalone executable, placing it on the desktop, and from there installing the Conduit software. The next step the user takes is probably to remove the cdtdeploy executable as he is hopefully happy with the installation of software. Without the cdtdeploy program available, the user can not remove software installed through Conduit if the uninstallation process is solely based on the cdtdeploy program.

With this problem in mind, the strategy of placing the uninstallation logic can become quite complex. The user should be able to remove software through add/remove, either with cdtdeploy present or not. One solution to the problem would be to create a Windows Installer package for the cdtdeploy software, to make sure that the user installs the cdtdeploy software as any other program. This would force the user to treat the cdtdeploy software as an ordinary installable software, and it would most likely keep the user from deleting it on a whim. It would also enable the custom action, started with add/remove, to get the location of the cdtdeploy through Windows Installer, and invoke it. The uninstallation process can then be based on invoking cdtdeploy from the add/remove location, and removing the program through cdtdeploy. In this scenario the user can still remove the cdtdeploy software and render the software installed uninstallable. To resolve this problem the custom action could start the uninstallation itself if it does not find cdtdeploy installed.

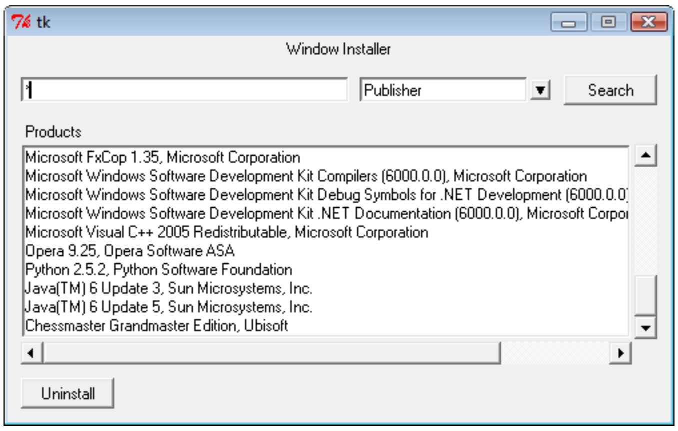
5.4.2 A better reality

As mentioned above the Windows Installer offers some complementary features that Conduit can use. One of these features is the Windows Installer database that holds track of installed software. Through the utilization of this database, Conduit could be able to recognize software that is already present on the user's computer.

Since Windows Installer is a very common way for installation of software on Windows, the list of installed software that windows installer has corresponds quite well with the actual installed software on the computer. This, with the fact that the list is easily available, could make Conduit able to see if required software of a release is already installed on the computer, and if so, keep from installing redundant software. This would be a improvement to the current system, where you for instance can install Python even though it is already installed through Windows Installer. With multiple Python versions installed on the computer, conflicts could arise surrounding what Python program is invoked when the user wants to start the Python interpreter.

To get this connection between the Windows Installer database and Conduit database, software in Conduit must be identified the same way as they are described by the ordinary installer. This is needed to set a proper relationship between software in Conduit, and software outside. For instance Python would need to identify itself as Python in Conduit, just as it would need to be described as Python in the installer provided by the creators of Python.

The implementation of such a feature can be based on the Python module pwis (see [17] for more details), which I created to list the installed software on a user's computer. This module consists of a shared library (.dll) that acts as a wrapper for some of the Windows Installer API functions. The shared library is used by a wrapper module through ctypes. The main module is an Installer class, which consist of methods to extract information from software installed through Windows Installer. Conduit can use this module to extract information from the installed software, compare it to the software in its database, and mark them as installed if the information corresponds correctly.



```
Microsoft FxCop 1.35, Microsoft Corporation
Microsoft Windows Software Development Kit Compilers (6000.0.0), Microsoft Corporation
Microsoft Windows Software Development Kit Debug Symbols for .NET Development (6000.0.0
Microsoft Windows Software Development Kit .NET Documentation (6000.0.0), Microsoft Corpo
Microsoft Visual C++ 2005 Redistributable, Microsoft Corporation
Opera 9.25, Opera Software ASA
Python 2.5.2, Python Software Foundation
Java(TM) 6 Update 3, Sun Microsystems, Inc.
Java(TM) 6 Update 5, Sun Microsystems, Inc.
Chessmaster Grandmaster Edition, Ubisoft
```

Figure 5.1: Showing pwis, a Python module that lists installed software that is installed through Windows Installer.

With the introduction of such a feature the Conduit system would expand its horizon by also supporting already installed software. This would allow Conduit to check for software that is not installed through Conduit. This would lead to less redundant software installed on the user's computer, and less traffic on the Conduit distribution system.

Bibliography

[1] Arve Knudsen. Generic software distribution and deployment. 31. January 2006.

[2] Mike Gancarz. The unix philosophy. ISBN1555581234.

[3] Microsoft. Windows installer. http://msdn.microsoft.com/en-us/library/cc185688(VS.85).aspx.

[4] Fourth Edition The American Heritage Dictionary of the English Language. Definition of integrate. http://www.thefreedictionary.com/integrated.

[5] Microsoft. Rollback installation. http://msdn.microsoft.com/en-us/library/aa371370(VS.85).aspx.

[6] Microsoft. Windows installer sdk. http://www.microsoft.com/downloads/details.aspx?FamilyId=6A35AC14-2626-4846-BB51-DDCE49D6FFB6&displaylang=en.

[7] Microsoft. Windows installer - database tables. http://msdn.microsoft.com/en-us/library/aa368259(VS.85).aspx.

[8] Microsoft. Windows installer - component table. http://msdn.microsoft.com/en-us/library/aa368007(VS.85).aspx.

[9] Microsoft. Windows installer - file table. http://msdn.microsoft.com/en-us/library/aa368596(VS.85).aspx.

[10] Microsoft. Wix. http://wix.sourceforge.net/.

[11] Acresso. Installshield. http://www.acresso.com/products/installation/installshield.htm.

[12] Python standard library. msilib. http://docs.python.org/library/msilib.html.

[13] Microsoft. Windows installer - installer function reference. http://msdn.microsoft.com/en-us/library/aa369426(VS.85).aspx.

[14] Mark Hammond. Python win32 extensions. `http://starship.python.net/crew/mhammond/win32/Downloads.html`.

[15] Thomas Heller. Ctypes. `http://python.net/crew/theller/ctypes/`.

[16] Microsoft. Automation interface reference. `http://msdn.microsoft.com/en-us/library/aa367810(VS.85).aspx`.

[17] Erik Fløisbonn. pwis - python module listing windows installer installed software. `http://heim.ifi.uio.no/erif/pwis/`.

List of Tables

List of Figures