Växjö University

School of Mathematics and Systems Engineering

Reports from MSI - Rapporter från MSI

XAP Integration

Qinghua Liu
Mingjie Zhu

Nov
2006

MSI
Växjö University
SE-351 95 VÄXJÖ

Report 06157
ISSN 1650-2647
ISRN VXU/MSI/DA/E/--06157/--SE

Abstract

This bachelor thesis will present the XAP tool integration project. Apart from presenting the survey of integration techniques that includes integration models and case tool models, we have conducted a comparison of these models. Then we reason about their applicability in the XAP setting. We apply this survey into the XAP tool integration project – integrate three tools in one IDE on data level. In this IDE, the user can create a new project and use these three tools freely in the new created project. The database among them is shared.

KEY WORDS: CASE CASE Technology CASE Tools XAP IDE Data Integration

Table of Contents

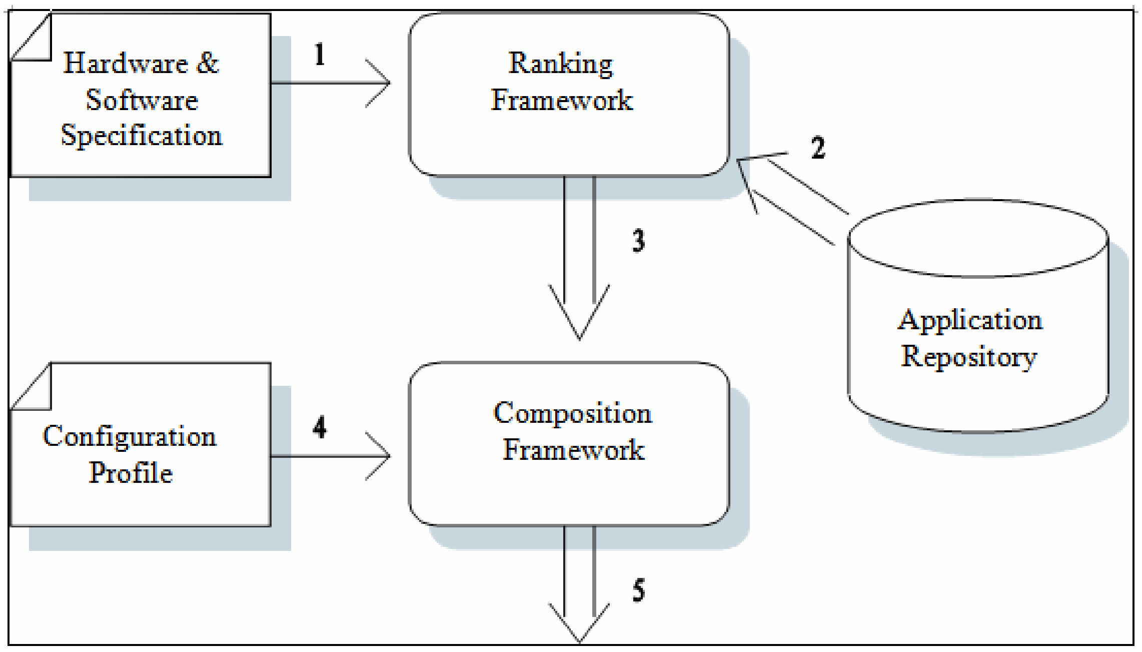
1. Introduction

With more and more softwares are published and involved in one system, integration becomes more and more important. We need run different softwares in one system, and apply them on one project. Different levels of integration such as tool integration, data integration, and process integration come out to meet the various need of integration.

This project is a toolset for the XAP system. It is to integrate available tools in IDE so that it will be easier to use these for XAP system. It is more focused on data integration level for each tool shared one database. This project is for bachelor thesis in Växjö University. In order to understand the project better, we first give some information of the XAP system.

1.1. Background

The eXtensible Application Provisioning (XAP) system is a framework and tool suite for software provisioning, e.g. "the capability to receive a request for an application, finding a suitable version of the requested application and provide it to the requestor". The processes supported by the frameworks include component selection (Ranking Framework, SF), composition support (Composition Framework, CF), and a storage facility for applications (Application Repository, AR). The AR contains descriptions of Application Families using a feature-tree notation, which makes it possible to configure the application in a flexible way. A short description of the interactions of these three components is as follows:



Here is a main description of workflow:

1. A hardware and software specification is sent to the RF. For instance, a user with a laptop wants applications to program with Java. So he asks XAP for help. He enters his laptop's hardware and software specifications such as how much memory he has and so on. These specifications will be sent to the RF.
2. The RF retrieves application description from the AR and performs a selection of applications that are suitable for the user according to the hardware and software specifications. For instance, AR sends the RF what compiles and editors it has. According to the specifications that stores in RF, RF selects a suitable compile and

editor for user. For sure the compile makes the user edit and compile most efficiently.
3. The selected applications' descriptions are sent to the CF. For instance, send the name and size of the editor and compile to the CF.
4. A configuration profile is sent to the CF as well. The configuration profile contains information about how the user wants the application configured.
5. The CF composes the application according to the application description.

If we take example of the mobile phone user, we will find that if the user wants to download a new game, then the hardware and software specification of his mobile will be sent to the RF, RF will retrieve these descriptions from the AR and do the selection. According to the selection and configuration profile, CF will compose the application which is the best choice for the user's mobile phone.

To configure the XAP system and the managed application families, we need to maintain and edit several configuration and data files. But it is very easy to make mistakes when somebody edits files directly. So there are some tools available to make these tasks easier. In order to use these tools expediently, we need to create a workbench where people can run these tools.

1.2. Problem Statement

The goal for this thesis project is to develop an integrated environment supporting the development, configuration, and maintenance of provisioning projects. Within XAP there are developed several different tools, all supporting some aspect or aspects of provisioning. These should be adapted and integrated into the environment. Now there are three tools available:

1. The first one is RepositoryEditor that will display all the applications that store in database. It is a SWT application. For each application, user can see components and some attributes of the components and application. The user can edit these attributes.
2. The second one is BuildApplicationGUI that is a SWING application. It allows the user to choose user agent, category and applications. The user may select, configure and compose these.
3. The third one is ConfiguratorGUI that also is a SWING application. It can be used to configure these three tools. In this application, you can set the workspace path and the database properties.

1.3. Project Delimitation

The delimitation of this thesis project is that the three tools to be integrated are not in the same type -- one is SWT and the other two are SWING. There is a technical problem with integrating these three tools.

1.4. Project Solution

The main delimitation of this project is how to handle two kinds of applications in one framework. As we know, till now, there is no method to integrate these two applications except that we rewrite one of the two or call one of the two outside our framework. The two methods have their own advantages and disadvantages. After comparing them, we decided to call one of the two outside our framework. In this way, the possibility of making mistakes decreased because we did not need to rewrite a lot of code which would have introduced more mistakes, and it also saved us a lot of time.

2. CASE

In this chapter, you will be presented with the survey of the CASE from CASE technology and CASE tools. Also four models of software development environments.

2.1. Introduction

CASE – Computer Aided Software Engineering has many definitions and no one can give a complete, clearly idea without argument. For example, Sodhi **[1]** gives the definition based on the production-process technology:

"Computer Aided Software Engineering (CASE) encompasses a collection of automated tools and methods that assist Software engineering in the phases of the software development life cycle."

Pressman **[2]** defines CASE as

"The workshop for software engineering is called an integrated project support environment, and the toolset that fills the workshop is CASE."

Fuggetta**[3]** presents the definition:

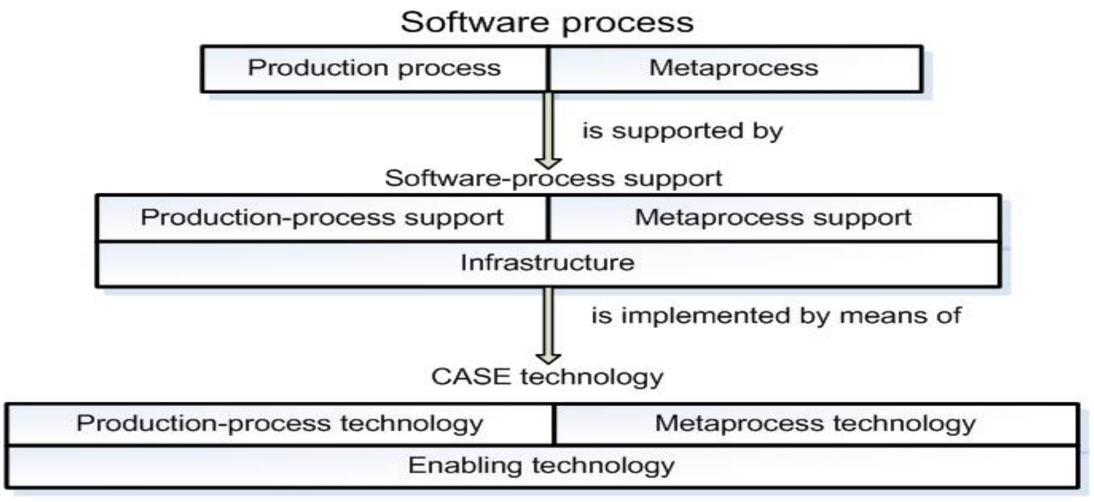
"CASE – computerized applications supporting and partially automating software-production activities."

On the surface, we can say CASE is the integration of tools and technology that supports a number of processes forming a methodology that will support the (parts of) software development life cycle. CASE technology is a software technology offering automation and engineering's guideline for software development, maintenance and project management. A CASE tool is the software tool which supports specific software life cycle automation (at least partly automation). CASE methodology is a kind of "automatable" methodology. It defines a strict, engineering process for the development and maintenance of software.

2.2. Classification of CASE Technology

Before the classification, first we need to know something about software process. According to the Conradi et al, the software process can be divided into two sub processes: a production process and a metaprocess. The production process includes all activities, rules, methodologies, organizational structures, and tools. It must be defined, assessed, and evolved through a systematic and continuing metaprocess. Metaprocess is to get acquisition and exploitation of new products supporting software-production activities and the improvement and innovation of the procedures, rules and technologies.

Meanwhile, it is implemented by means of CASE technology which contains production-process technology, metaprocess technology and enabling technology. The following picture shows the relationship among them.



The classification is based on this general framework. And following is the three CASE tool categories classified in the production-process technology:

A production process may be viewed as a set of elementary tasks to be accomplished to produce a software application. Tasks are grouped to form activities. For example, coding is an activity that includes editing, compiling, debugging, and so on. Meanwhile, editing, compiling, and debugging are tasks.

- Tools: support only specific tasks in the software process.
- Workbenches: support only one or a few activities.
- Environments: support (a large part of) the software process.

Tools can be thought single products or the components of the workbenches and environments.

2.2.1. Tools

They are the most elementary types to support the specific tasks in the software process.

For example, here are some tools:

Editing tools like traditional test editors and word processors. They can be used to produce textual documents, textual specifications, and documentation.

Programming tools like some traditional compilers. They are used to compile.

2.2.2. Workbenches

They integrate tools supporting specific software-process activities in a single application.

They should have a homogeneous and consistent interface. They should call other tools easily. They also should have an access to a common data set.

Here are some workbenches as examples:

Project is a workbench that makes projects managers control their project in a good way.

Open workbench is an open source windows-based desktop application. It provides robust project scheduling and management functionality.

2.2.3. Environments

Environments are the collections of tools and workbenches which support the software process.

Here are some environments for example:

Eclipse is an open source environment. It provides an extensible development platform and application frameworks for building software. Eclipse based tools give developers freedom of choice in a multi-language, multi-platform, multi-vendor environment. It includes a lot tools to support the software process like smart editor, compiler and so on.

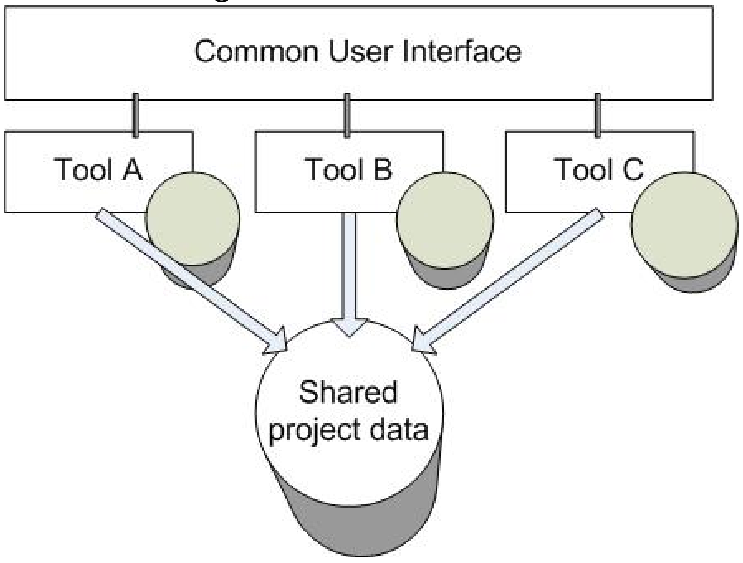
Visual studio provides a range of tools that offer many benefits for developers. It makes the software process easier. It includes a lot tools as well, like debugger.

2.3. Three functional areas of CASE Tool Integration

CASE tool is to support specific software development and since some tools share a similar user interface or a vendor offers several tools to support more than one phase of the software development life cycle, we need to integrate these tools. That is we called CASE Tool Integration.

Analysis of CASE tool integration is generally separated into three functional areas: data, control, and presentation integration.

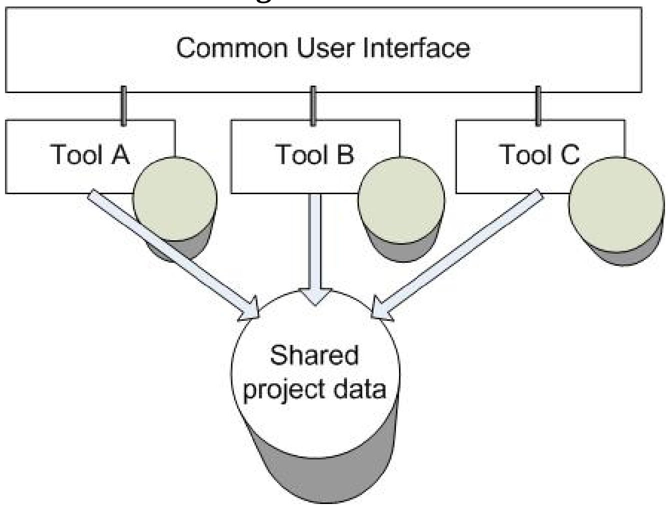
2.3.1. Data Integration



Data integration is also called data sharing. The goal of data integration is to ensure the information is consistent without the reflection of other tools' operation or transformation.

For example, there are several applications that are using separate databases. To integrate these applications in data level means that integrator should make these applications share a common database in which all applications deposit their data. That is to make these applications access the database through the same schema. Consider the editing tool, building tool and compiling tool, they operate on different files. By integration them, these three tools should be able to operate on the same file.

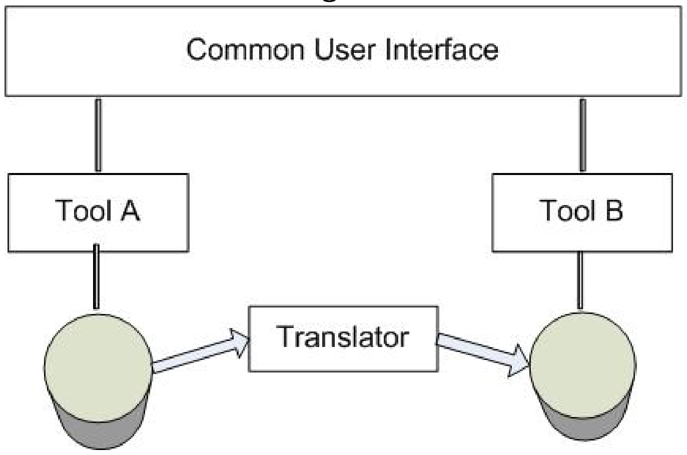
2.3.2. Control Integration



The goal of control integration is to admit the flexible combination of the environment's functions, according to the project preferences and the underlying processes the environment supports. It is about the tool communication and interoperation.

For example, when an editing tool announces that changes have been made to a source file, a building tool may receive this information and initiate a new system build. What more is that one tool may directly request that another tool perform an action by sending it a control signal. For example, the building tool may request that the source file to be compiled by a particular compiling tool.

2.3.3. Presentation Integration



The goal of presentation integration is to improve the efficiency of the user's interaction with the environment by reducing the user's cognitive load.

Still use editing tool, building tool and compiling tool as example. They have the different styles of interface to user. This is not convenient. By integration them on the presentation level, they can have the same style of interface. Definitely it will improve the efficiency of the user's interaction with the environment. Maybe these tools can be some buttons like in eclipse and visual studio.

In some thoughts, they analyzed the CASE tool integration into four dimensions, which are data, control presentation and process. Here we put the process integration together with the presentation integration.

2.4. Five areas of CASE Tool Integration

As it is referred in former section, the vendor's decision will affect the CASE tool integration. So in this section we examine five areas of CASE tool integration from the perspective of the end-user.

2.4.1. Single-vendor tool integration

It is the first CASE tool integration. As its name, this integration faces to the tools and data of a single vendor.

- **Problems**:

1. limit range of tools

This means that the user can only use the tools provided by the particular vendor. Most of CASE vendors do not provide all the tools that are needed in a full life-cycle of development. Because of this limitation, many users think it as an unacceptable solution.

2. data content consistency

This is caused by the fact that some tools use localized data dictionaries to store tool objects. The end-user should take care of the data synchronization and reconciliation.

3. impact the flexibility or modification

If the user can modify the interface or objects of such a nonstandard toolset, the compatibility is a problem even if conversion capabilities are developed to allow a database or interface to be adapted to a standard format. But this work will be done by the user.

4. use of relational database

Due to limitations of the relational model, relational database can not readily accommodate the flexible, complex object/data types. Actually the problem is a function of data size relative to performance characteristics. That is because relational databases store data by type not by the relationships within the data. So when the CASE required a large amount of data will increase the amount of time required to process database transactions.

- **Solutions**:

1. change the scope of the problems

Some vendors tried to get leverage from each other's tool offerings and to fill in the technology/product gaps. But at last, it fails. The limits or gaps are still there.

2. share database

This lets some tools to share data dictionaries. It can solve the problem more or less. But if not implemented carefully, it will affect the performance profile of the toolset. But anyway it provides a common basis for internal tool integration and a means of transition to external one.

3. ORDB

An object-relational database is a relational database management system that allows developers to integrate the database with their own custom data types and methods.

2.4.2. Multiple-vendor tool integration

This integration is a kind of external integration. It has two forms: access control in which the vendor allows the tools to be invoked by other vendors' tools and data control in which the vendor allows other external tools operate on internal dictionary.

- Problems:

1. data incompatibility

In order to help other vendors integrate their own tools with the vendor tools, many vendors publish their tool/data interfaces to others. But the data is still not fully described. The format of the data can be different. The contents of the dictionaries may be inconsistent.

Some tools can be imported or exported to deal with the problems of data dictionaries that can not be shared by different tools. But there is a problem of data loss/mismatch when attempting to merge two incompatible data sets.

2. selection and acceptance of standards

To integrate another vendor’s tool/data interfaces is very expensive. The cost depends on the number of nonstandard interfaces and the number of platforms and environments supported by these other vendors.

3. coordinate changes manually

The user may have to write code to integrate a set of vendor tools if the user can expend the time/cost to implement the tool integration. During this phase, users should coordinate changes by himself.

- Solutions:

1. Define a central data repository

These kind data repository would take steps to solving the data incompatibility issue. A repository is a common shared database which stores the rules associated with CASE data objects. The data may reside in the repository or may be contained in local databases.

A central data repository makes the content of the dictionary consistent. The problem of different formats of data can be resolved. Because now by using one repository, the data formats are the same.

2. Form strategic partnerships, mergers or acquisitions

Some vendors are teaming up to add to their existing tool offerings and to attempt to affect a CASE tool standard.

2.4.3. Operating environment integration

It includes both the base computing environment and the add-on tools and utilities. Some tools are for personal computers while some are for workstations or mainframes. Which system to be used depends on the technical or commercial application to be hosted by the toolset.

- Problems:

1. Tools tied to environment

These tools use specific features of the operating system or support toolset that are unavailable on other systems. It is hard to host these tools to other environments.

2. Scalability

Some tools can not perform all their features because of environment limitations. For instance, tools developed for a single-user PC environment may not support multi-user projects.

3. Configuration management system

Extension of CASE tools to include configuration support must be given careful consideration in order to prevent the scalability problems.

- Solution: standardized middleware

Middleware provides a mechanism that allows one entity to communicate with another entity or entities. A middleware product can be installed in different environments. It is just a tool, nothing more. Why use it? The answer is simple: because it really is the best and only solution that allows applications to communicate with one another. It hides the complexities of internal fact by giving a simple middleware API. The users do not need to consider the scalability problems.

2.4.4. Development process integration

CASE tools now can help in the whole process of software development. It is no longer facing certain phase. It can be anticipated of smoothing process transitions.

- Problem

1. Tool flexibility

Users want the tools to be modified and adapted easily. But most CASE tools can not. Some others just allow the user to modify on the presentation level. In order for CASE tools to be fully integrated, users must be able to modify the characteristics behavior of the tool.

2. Support methodologies

Most CASE tools support only very specific design methodologies. The methodology supported by the tools is usually strictly enforced. In order to accommodate methods, CASE tools should be able to expand beyond the traditional “bottom up” or “top down” design approaches.

- Solution

Some large systems/CASE vendors create several comprehensive methodology/toolset offerings. Some tools are developed to support these methods. Each will also offer the consistency of a central data repository. Each vendor is trying to define the standard by default to make as many users/vendors behind their products as possible.

Because of the emergence of CASE and the demands, a new emphasis has been come out. Incorporation of all aspects of the development life cycle into a seamless product line is needed. Cooperation is necessary. IBM, DEC are now likely to end up supporting a common IPSE standard.

I-CASE has the similar situation with IPSE, but is targeted more at the commercial marketplace. Its emphasis is on the integration of the front-end phases of development life cycle which lead to support of code generation.

Framework provides a tool management executive which handles the overhead involved with coordination of the tool suite.

2.4.5. End-user integration

- Problem

A consistent user interface/presentation is the problem in general. CASE adoption needs a significant investment for organization. The cost is more than the cost of the tools. It needs time, training and support.

- Solution

A standardized user presentation format can help to decrease the cost to learn and provide smoother transition between different vendor offerings. When selecting of a standard, commonality must be considered.

User interface consistency must be paid attention to.

2.5. Models of Software Development Environments

The model of software development environments consists of three interrelated components: policies, mechanisms and structures.

Policies are the requirements imposed on the user of the environment during the software development process.

Mechanisms are the languages, tools, and tool fragments that operate on the structures provided by the environment and that implement the policies supported and enforced by the environment.

Structures are those objects or object aggregates on which the mechanisms operate.

Considered the effect of scale of the system, there are four classes of models that emphasizes on the scale of the system which are individual, family, city and state models. The individual and family models represent the current state of the art, although they are not suitable for the large systems. While, the city and state models are better but need more research on them. In the following, these four classes will be presented with three components which are policies, mechanisms and structures.

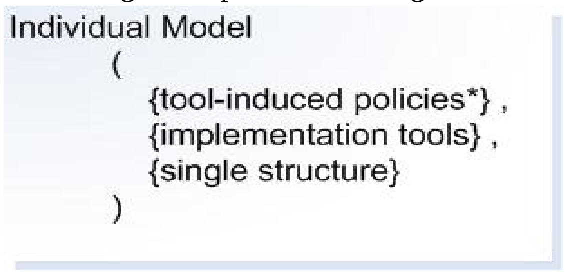
2.5.1. Individual Model

Construction is the dominative issues of this class of models. This model of environments is often referred to as programming environments.

The policies are laissez-faire and hard-wired. Tools provided by mechanisms are editors, compilers, debuggers and so on. And the environments often provide a single structure.

Mechanisms dominate in this class. Mechanisms lead on the policies and instruct structures to make the tools work together.

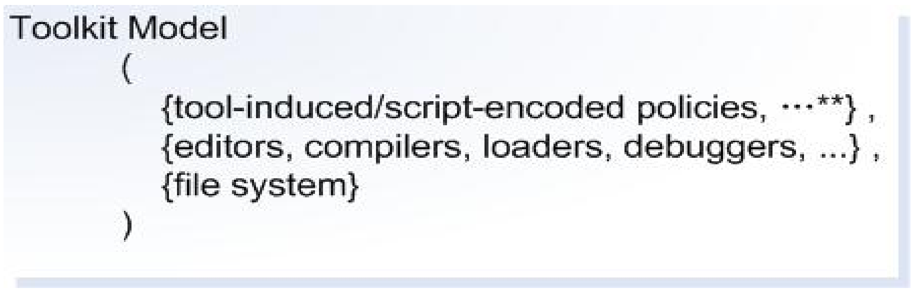
Following is the picture showing the three components of this model:



Individual Model
(
{tool-induced policies*} ,
{implementation tools} ,
{single structure}
)

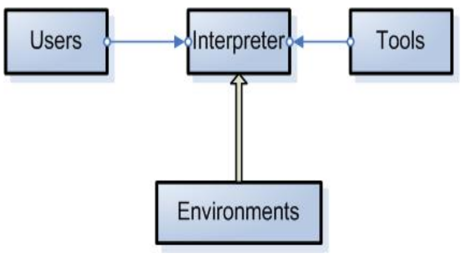
To instantiate this model, there are four groups of environments presented: toolkit environments, interpretive environments, language-oriented environments and transformational environments.

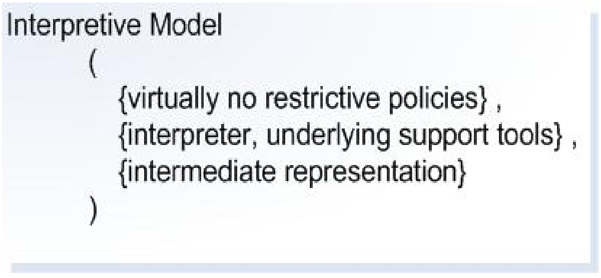
Toolkit environments are the prototype of the individual model. The policies of this model are weak and the communication among mechanism tools just through the file system.



Toolkit Model
(
{tool-induced/script-encoded policies, …**} ,
{editors, compilers, loaders, debuggers, ...} ,
{file system}
)

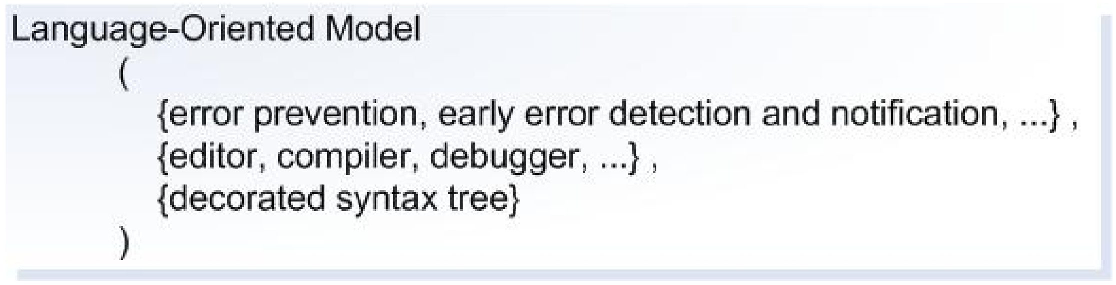
Interpretive environments are also the early version of the individual model. But in this model, there is an interpreter between tools and users which use a single language as the interface. While the language is still depend on the environments. The structure is an intermediate representation instead of the file system. The policies is also different from the toolkit one, there are virtually no restrictive policies that force the programmer.





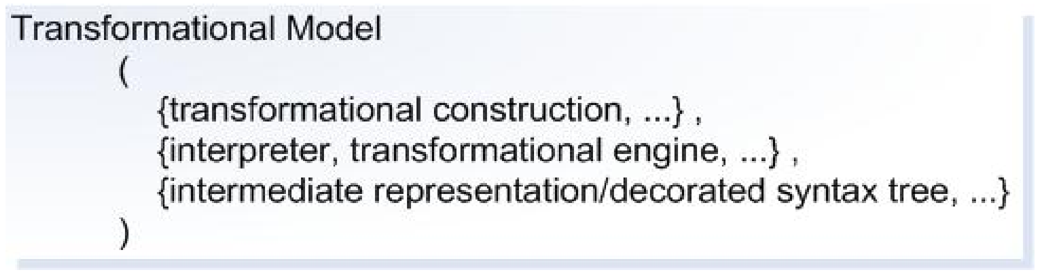
Interpretive Model
(
{virtually no restrictive policies} ,
{interpreter, underlying support tools} ,
{intermediate representation}
)

Language-oriented environments mix the toolkit and interpretive models. There is an important policy in this model which is early error detection and notification. Tools in this model try to keep consistency between input and output. And the structure is no longer a simple structure; it is a decorated syntax tree.



```
Language-Oriented Model
        (
           {error prevention, early error detection and notification, ...} ,
           {editor, compiler, debugger, ...} ,
           {decorated syntax tree}
        )
```

Transformational environments support the wide-spectrum language. And the mechanisms are the transformational engine and the interpreter. The structure is a cross for intermediate representation and decorated syntax tree.



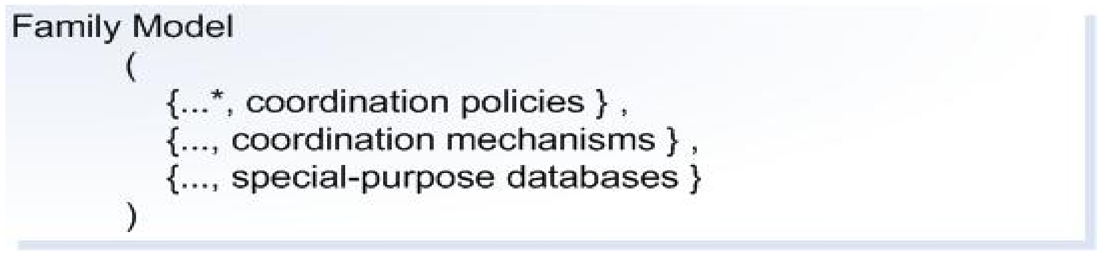
```
Transformational Model
        (
           {transformational construction, ...} ,
           {interpreter, transformational engine, ...} ,
           {intermediate representation/decorated syntax tree, ...}
        )
```

2.5.2. Family Model

Family model can represent the current state of art in the software development environments. Generally speaking, this model is an extension of individual model. The most difference between individual and family model is the interactions of programmers. Reacting in the policies, some rules are needed for the interaction in the family model. And it also extends the enforced coordination which is about version control and configuration management in the mechanisms and structures.

In this class of models, coordination is the dominative issue. And the structures dominate in this class.

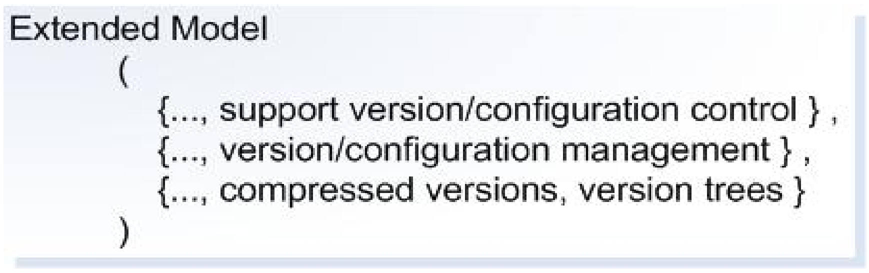
Following is the picture showing the three components of this model:



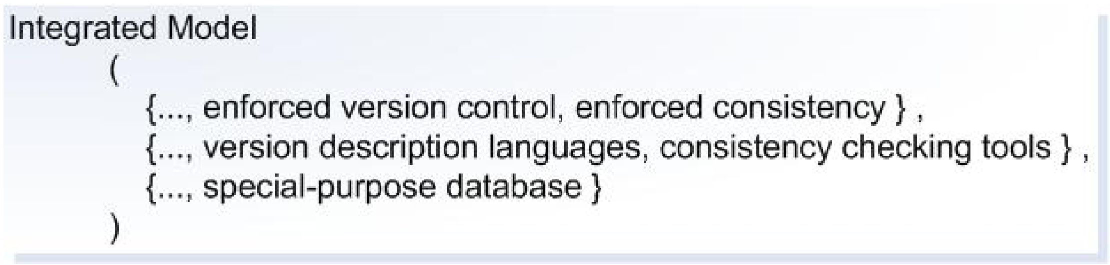
```
Family Model
        (
           {...*, coordination policies } ,
           {..., coordination mechanisms } ,
           {..., special-purpose databases }
        )
```

There are four groups of environments for instantiating this model, which are extended toolkit environments, integrated environments, distributed environments and project management environments.

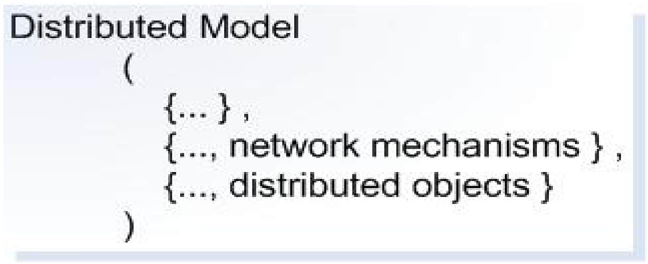
Extended toolkit model adds the configuration control and version control structure in the mechanism component part of the individual toolkit model. The basic mechanisms in the individual toolkit are preserved. And coordination between programmers are supported.


```
Extended Model
        (
           {..., support version/configuration control } ,
           {..., version/configuration management } ,
           {..., compressed versions, version trees }
        )
```

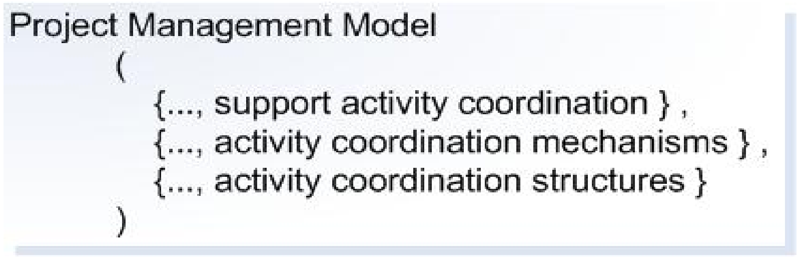
Integrated model is the extension individual language-oriented model. It preserves the consistency policy and enhances it not only among the modules but also within a module. And the structure is a special-purpose database.


```
Integrated Model
        (
           {..., enforced version control, enforced consistency } ,
           {..., version description languages, consistency checking tools } ,
           {..., special-purpose database }
        )
```

Distributed model adds the concept of network into integrated model. Additional structures are required for a number of machines connected and provide the reliability and high availability.


```
Distributed Model
        (
           {... } ,
           {..., network mechanisms } ,
           {..., distributed objects }
        )
```

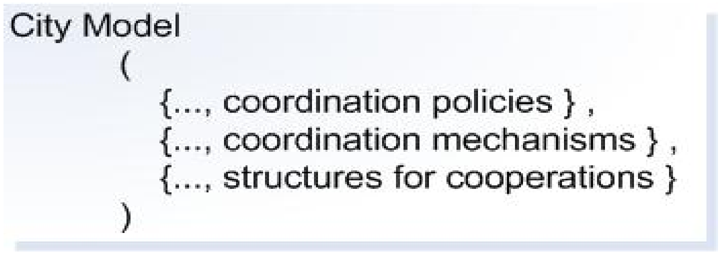
Project management model provides the mechanisms and structures of assigning tasks to individual programmers to support the coordination activities.


```
Project Management Model
        (
           {..., support activity coordination } ,
           {..., activity coordination mechanisms } ,
           {..., activity coordination structures }
        )
```

2.5.3. City Model

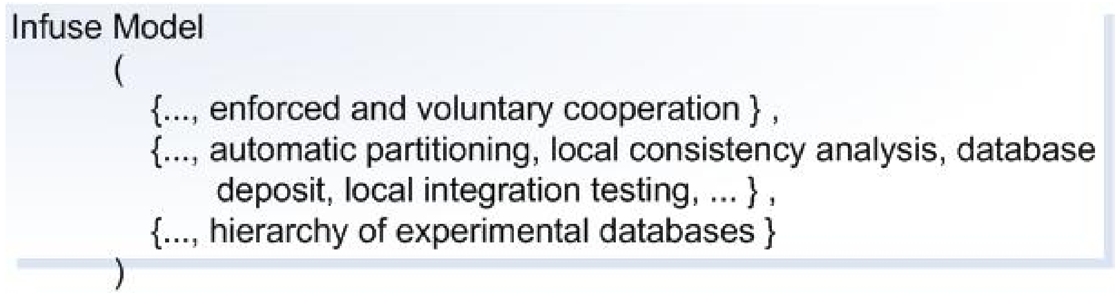
With the size of the project grows, the policies in family model is no longer sufficient. Coordination should be the dominative issue in this class of models and the policies should dominate this class.

Following is the picture showing the three components of this model:


```
City Model
        (
           {..., coordination policies } ,
           {..., coordination mechanisms } ,
           {..., structures for cooperations }
        )
```

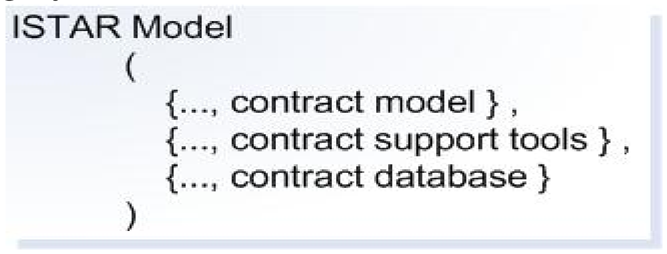
The family model does not have the adequate rules for a large increase in scale, and we really need a model like city model which concerns cooperation policies. But little work is focus on this model and we have only two environments for instantiating this model, which are Infuse and ISTAR.

The primary concern of Infuse is the technical management of evolution in large systems. It is about what kinds of policies, mechanisms and structures are needed to automate support for making changes by a lot of programmers.



```
Infuse Model
        (
           {..., enforced and voluntary cooperation } ,
           {..., automatic partitioning, local consistency analysis, database
                 deposit, local integration testing, ... } ,
           {..., hierarchy of experimental databases }
        )
```

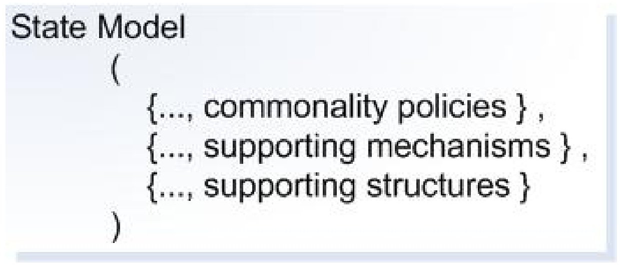
ISTAR is about the managerial problems of managing system evolution. It provides an environment for managing the cooperation of large groups of people producing a large system.



```
ISTAR Model
        (
           {..., contract model } ,
           {..., contract support tools } ,
           {..., contract database }
        )
```

2.5.4. State Model

In order to reduce in cost and improve in productivity, we need state model. Reuse of various kinds is possible: tools may be distributed with little difficulty; code may be reused; design and requirements may be reused.



```
State Model
        (
           {..., commonality policies } ,
           {..., supporting mechanisms } ,
           {..., supporting structures }
        )
```

In this model, the concern for commonality, for standards, is dominant. This policy will induce policies in the specific projects. So the state model’s policies are higher-order policies because they can induce policies.

2.6. Summary

In this chapter, we discuss CASE from two parts which are CASE Technology and CASE Tool. First, we discuss the classification of CASE technology which is based on a general framework. According to this, CASE products have been classified into three categories: tools, workbenches and environment. Then we focus on the CASE tool part, we first discuss it into three functional areas which are data integration, control integration and presentation integration. Then we discuss it from the perspective of the end-user and list five areas of the integration which are single-vendor tool integration, multiple-vendor tool integration, operating environment integration, development process integration and end-user integration. After these, we also talk about four classes of models of software development environments which are individual model, family model, city model and state model.

Tasks are grouped to form activities, and sets of activities supporting parts of the software process. Normally we use one tool to support one specific task, like editing tool, compiling tool and debugging tool. And for one activity, it is composed of one or more tasks. For example, the coding activity, we need to integrate editing tool, compiling tool and debugging tool into one workbench. Furthermore, if we could integrate coding and testing activities into one environment that will achieve parts of the software process.

The classification of CASE Technology clarifies the different level in the software process, and the CASE Tool integration let us form the tasks to activities that will help to support parts of the software process. That is why we focus two sections on CASE Tool integration. CASE tool integration is really helpful to coordinating tools and data from a variety of sources and venders. But there is still no official standard for the CASE tool integration.

The taxonomy of the models of software development environments delineate the classes of interaction that the software developers use and indicate which part of the software development environment is need to be made more effort.

In this thesis project, we focus on functional areas of CASE tool integration instead of the perspective of the end-user, because the vendor of the tools to be integrated is the same and we do not need to worry about that.

3. Requirements

In this thesis project, we need to integrate three tools into one IDE on data level. The three tools are RepositoryEditor, BuildApplicationGUI and ConfiguratorGUI.

1. RepositoryEditor will display all the applications that store in database. It is a SWT application. For each application, user can see components and some attributes of the components and application. The user can edit these attributes and save them.
2. BuildApplicationGUI is a SWING application. It allows the user to choose user agent, category and applications. The user may select, configure and compose these.
3. ConfiguratorGUI also is a SWING application. It can be used to configure these three tools. In this application, user can set the workspace path and the database properties.

Besides integrating these three tools and letting them sharing one database, the two base non-functional requirements are flexibility and extensibility.

3.1. Flexibility

Our project is to integrate three tools in a workbench. While using this workbench, the user can create a new project by giving it a name and a short description.

In the workbench, flexibility means the user can run these three tools easily and create and open projects easily.

There are two features of our project to provide the flexibility. One is shared database, the other is that the user should have to either create a new project or open an existing project.

Sharing database means all the three tools use the same database schema. So we just need configure the database once, then the three tools can access to database without doing configuration thing.

Every project will have a file to store its information like name, description and date. The format of the file is simple. So that people even can create the file by him own. When creating a new project, the name should not be null.

3.2. Extensibility

Since there will be more tools integrated in the workbench, extensibility means that it will be easy to add some new tools to workbench. During design and implementation, we should take into consideration future growth so that the workbench can be expanded with new capabilities without having to make major changes to the system infrastructure.

4. XAP IDE

After doing some research on the three tools, we decide to integrate these tools on data level. Since there is no interaction between these three tools, there is no need to take the control integration. And presentation integration is optional, but thinking about the time and budget of the project, we just skip it. So, we follow the data integration, all the three tools share the same database, all the information in the environment is managed as a whole, regardless of which tool is used.

According to the classification we referred in the former chapter, in this project it is the workbench.

"They integrate tools supporting specific software-process activities in a single application."

"They should have a homogeneous and consistent interface. They should call other tools easily. "

"They also should have an access to a common data set."

We follow these standards to integrate the tools into one application, and after creating a new project, it is easy to call any tools and they share a common database.

Because in this project construction is the dominative issues, we choose individual model as our software development environment. In this class model, there are still four models which are toolkit model, interpretive model, language-oriented model and transformation model. But since we use the file system as the structure, we follow the toolkit model.

4.1.1. Integration of applications

The three tools are integrated into the one toolset and we talk some more detailed in this section.

4.1.2. Integration of RepositoryEditor

- Characteristics of RepositoryEditor: a SWT application which connects to database and gets data. Each application consists of several components. The structure displays with a tree. The root stands for application while the leaf stands for component.
- Integration strategy: it has the same type as our workbench. So we call and embed this application in our workbench.
- Integration problems: where to display RepositoryEditor's own menu.
- Solutions to integration problems: when user selects to run RepositoryEditor, its menu will be added to the framework's menu bar.
- Alternatives to selected solutions: to make a small window when RepositoryEditor is called. But it demands to code a lot, and it is not convenient as the one we use.

4.1.3. Integration of BuildApplicationGUI

- Characteristics of BuildApplicationGUI: a SWING application which connects to database and gets data. Different user agents have different categories. Different categories have different applications. The information is displayed in three dropdown menus. While selecting one from certain menu, corresponding information will be displayed. After the three kinds are decided, user can select, configure and compose them.

- Integration strategy: it has the different type with our workbench. So we call this application outside our framework.
- Integration problems: the relationship between our framework and BuildApplicationGUI.
- Solution to integration problems: our framework is BuildApplicationGUI's father framework. It means that only after user exits BuildApplicationGUI our framework will activate.
- Alternatives to selected strategies: BuildApplicationGUI and our framework are at the same level. Then user can operate on framework. It does not show that BuildApplicationGUI is called by the framework.

4.1.4. Integration of ConfiguratorGUI

- Characteristics of ConfiguratorGUI: a SWING application which connects to database and gets data. It displays the information about username, password, port and host of database, and workspace as well. The information can be saved or loaded.
- Integration strategy: it has the different type with our workbench. So we call this application outside our framework.
- Integration problems: the relationship between our framework and ConfiguratorGUI
- Solution to integration problems: our framework is ConfiguratorGUI father franework. It means that only after user exits ConfiguratorGUI our framework will activate.
- Alternatives to selected strategies: ConfiguratorGUI and our framework are at the same level. Then user can operate on framework. It does not show that ConfiguratorGUI is called by the framework.

4.2. Use case specification

Attribute	**Value**
Use Case Name:	Create a project
ID:	1
Primary Actor:	End user
Brief Description:	The end user chooses to create a new project.
Pre-conditions:	The workbench starts
Post-conditions:	A new project is created successfully.
Normal Flow of Events:	1. End user selects "new project" item from "project" menu. 2. A dialogue box displays. 3. End user enters the name and description of the new project. 4. Give the name of the information file and specify path. 5. The specified project is created and information displays in the project information table.
Sub-flows:	3a. The name of the project is missing, application gives the error message.

Attribute	Value
Use Case Name:	Open a project
ID:	2
Primary Actor:	End user
Brief Description:	The end user chooses to open an existing project.
Pre-conditions:	The workbench starts and a project has already existed.
Post-conditions:	The existing project opens successfully.
Normal Flow of Events:	1. End user selects "open project" item from "project" menu. 2. A dialogue box displays. 3. End user selects an information file with extension "liu". 4. Information about the project displays in the project information table.
Sub-flows:	

Attribute	Value
Use Case Name:	Run RepositoryEditor
ID:	3
Primary Actor:	End user
Brief Description:	Run RepositoryEditor and make configuration.
Pre-conditions:	The workbench starts.
Post-conditions:	RepositoryEditor displays.
Normal Flow of Events:	1. End user selects "RepositoryEditor" item from "Tools" menu. 2. RepositoryEditor displays on the right side of workbench. 3. End user operates on RepositoryEditor.
Sub-flows:	2a. Database server is closed, RepositoryEditor does not run.

Attribute	Value
Use Case Name:	Run BuildApplicationGUI
ID:	4
Primary Actor:	End user
Brief Description:	Run BuildApplicationGUI and make configuration.
Pre-conditions:	The workbench starts.

Post-conditions:	BuildApplicationGUI displays.
Normal Flow of Events:	4. End user selects “BuildApplicationGUI” item from “Tools” menu. 5. BuildApplicationGUI displays outside workbench. 6. End user operates on BuildApplicationGUI.
Sub-flows:	2a. Database server is closed, BuildApplicationGUI does not run.

Attribute	**Value**
Use Case Name:	Run ConfiguratorGUI
ID:	5
Primary Actor:	End user
Brief Description:	Run ConfiguratorGUI and make configuration.
Pre-conditions:	The workbench starts.
Post-conditions:	ConfiguratorGUI displays.
Normal Flow of Events:	7. End user selects “ConfiguratorGUI” item from “Tools” menu. 8. ConfiguratorGUI displays outside workbench. 9. End user operates on ConfiguratorGUI.
Sub-flows:	2a. Database server is closed, ConfiguratorGUI does not run.

GUI description:

In our application, one tool are SWT, other two tools are SWING. It will not be a problem when we try to integrate the SWT tool in our application. But it became a problem when we try to integrate the two SWING tools in our application. The problem is that SWING component can not be displayed in SWT framework. Thinking about that, there are two choices for us to resolve this problem. One is to call the SWING tools outside our application; the other is to rewrite the SWING tools to change them to SWT tools. The first one is easier. But it requires data sharing that means the two SWING tools share the same database as the SWT tool. The second one needs a lot of work on the implementation that change SWING to SWT. At last, compare these two means; we decide to use the second alternative. It will be more extensibility when some other SWING tools need to be integrated. The integrator does not need to make a lot changes to system infrastructure. What needs to be done is to add the tool on the menu and call it outside.

The framework of our workbench has three parts.

One is the menu bar which contains two menus: project and tools. In the project menu there are two menu items; create project and open project. These two menu items fulfill use case 1 and use case 2. Some more menus will be added on when certain tool runs.

The second part is the project information part. It consists of a table. The table contains some information about what the name of the project is and when the project was created.

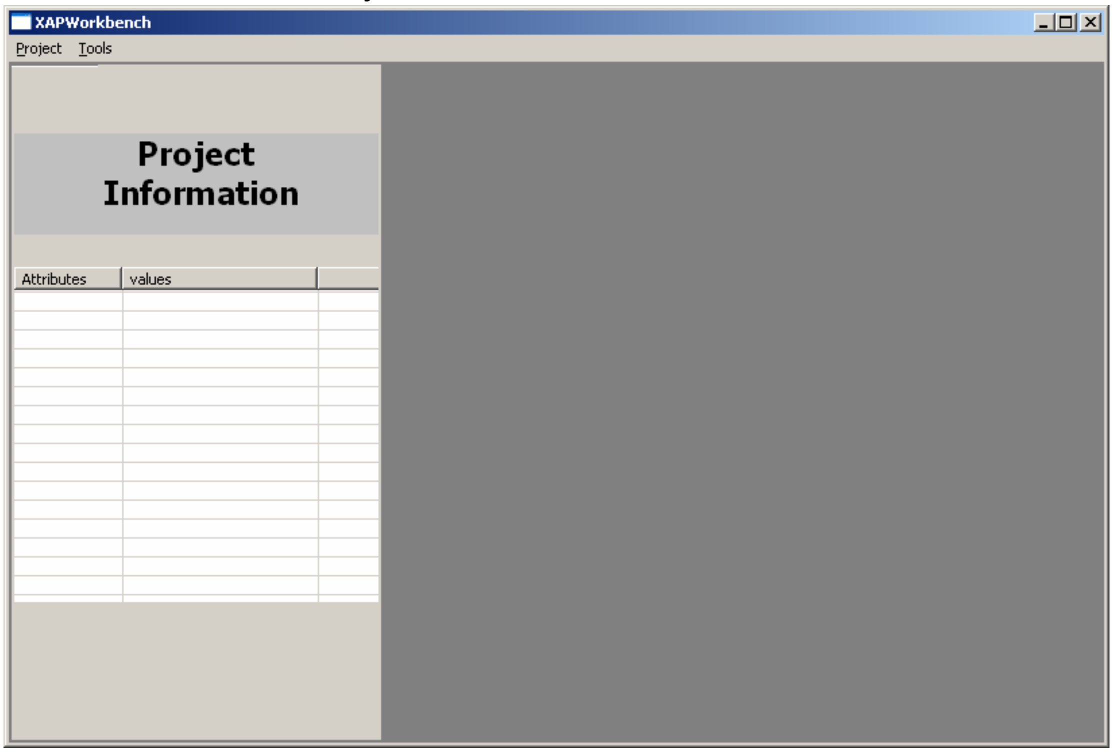
The third part is the place for tool running. As we said before, SWT tool runs inside the workbench while SWING tools run outside the workbench.

5. Evaluation

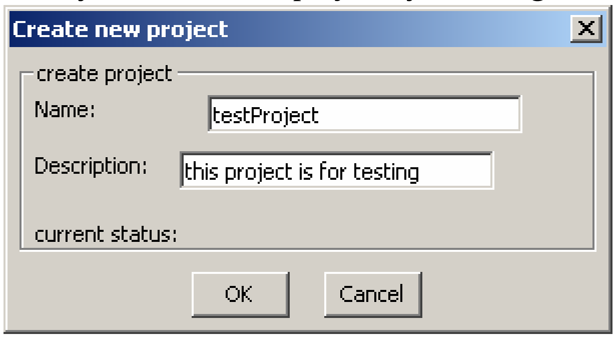
When our framework is called, it will connect to the database once. So we do not need to connect to database while calling different applications. This is because that the three applications share the same database schema. It is extendable because if a new application which is SWING will be added to our framework what we need to do is to call the new application and make it run outside. If the tool is SWT, we can call the tool inside the workbench and display it as part of the workbench. It is very easy to do that.

Our framework's interfaces are easy to understand. You just need to follow the instruction. Then you can achieve what you want. To prove how flexible it is we are going to create a new project called testProject and configure some date.

First we run the workbench you can see the framework, it looks like this:

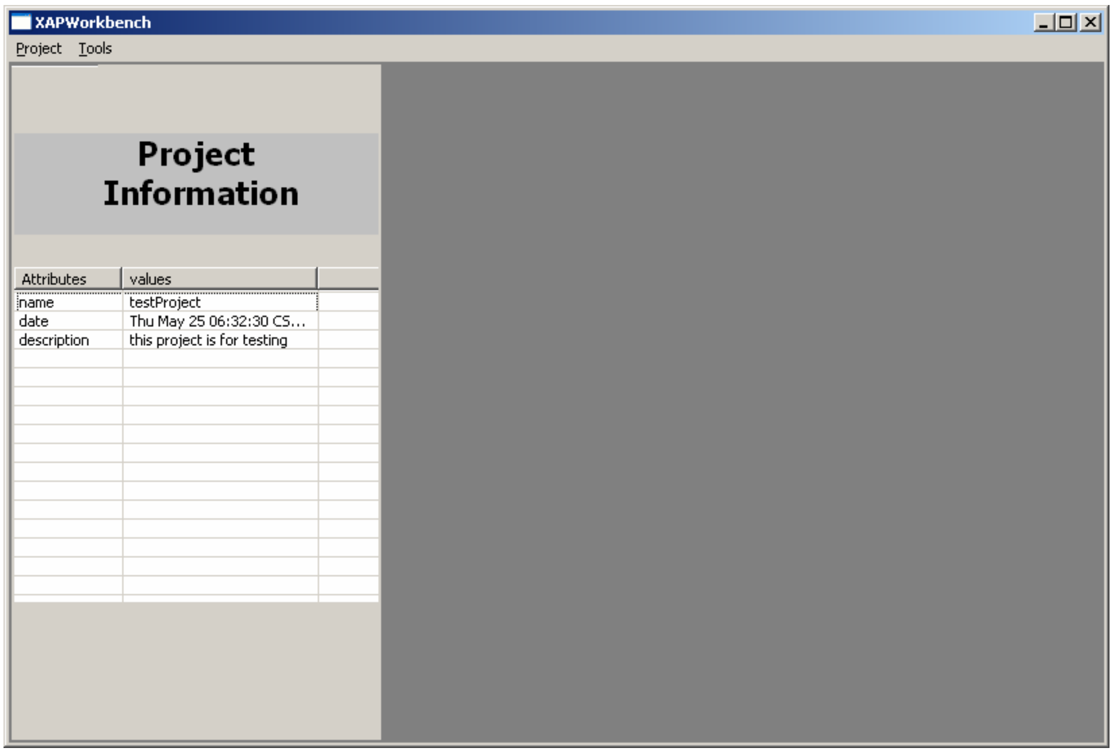


Then you can create a project by choosing the menu item "new project":



Name: testProject
Description: this project is for testing
current status:

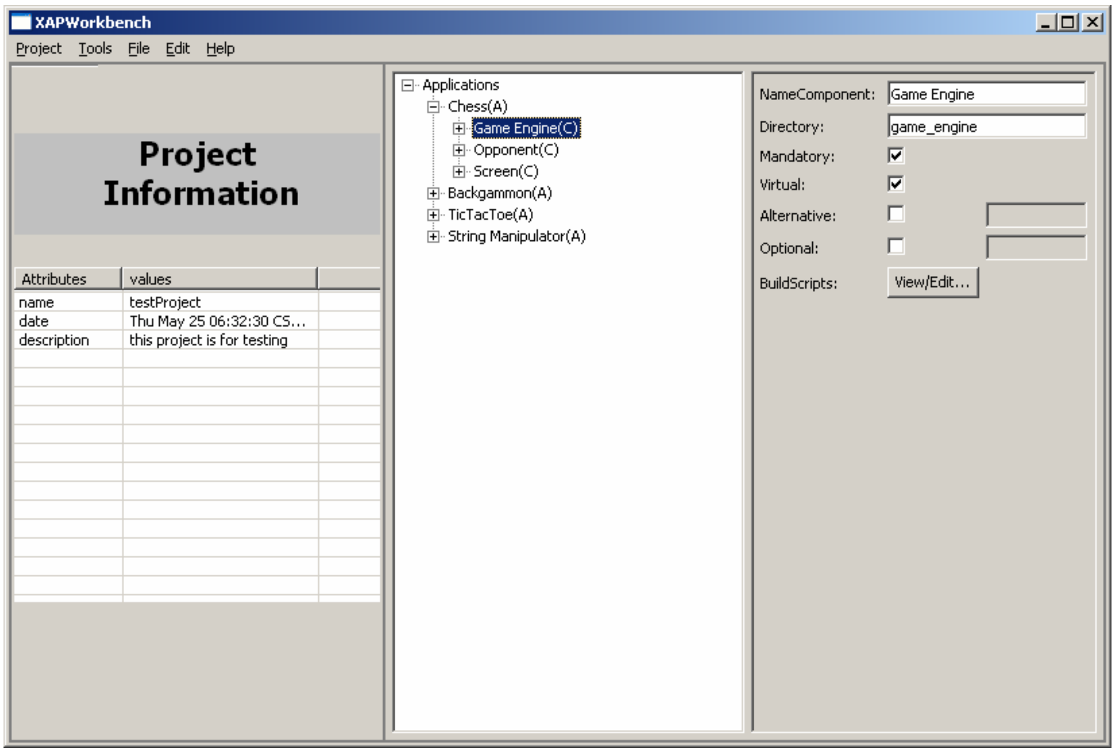
Then you can type the name and description of the new project. We set the name to be testProject. After that we give the name of the project information file and its path. Then you can see project information on the table:


XAPWorkbench

Project Information

Attributes	values
name	testProject
date	Thu May 25 06:32:30 CS...
description	this project is for testing

We can choose one of three tools like repository editor. It will display on the right of the workbench. We change the name of one component called Game Engine. Then we save it.


XAPWorkbench

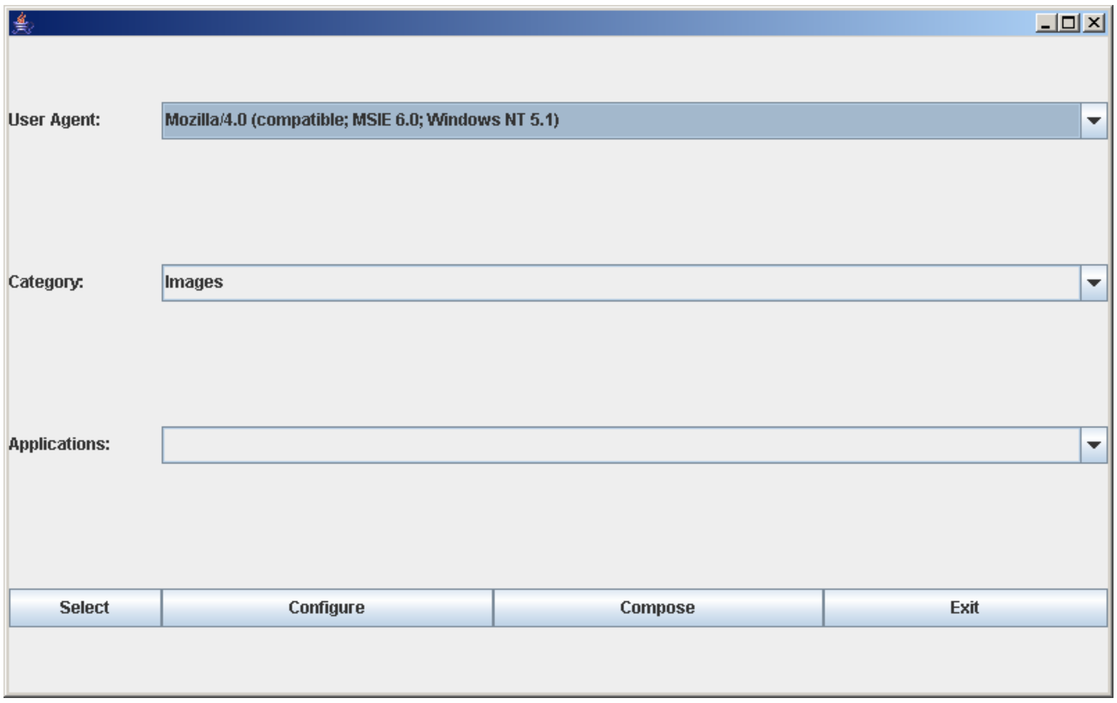
Project Information

Attributes	values
name	testProject
date	Thu May 25 06:32:30 CS...
description	this project is for testing

```
Applications
  Chess(A)
    Game Engine(C)
    Opponent(C)
    Screen(C)
  Backgammon(A)
  TicTacToe(A)
  String Manipulator(A)
```

NameComponent:	Game Engine
Directory:	game_engine
Mandatory:	✓
Virtual:	✓
Alternative:	☐
Optional:	☐
BuildScripts:	View/Edit...

Next we choose buildapplicationGUI tool. Here we can select different user agent with different category. Then we can compose them as you want.

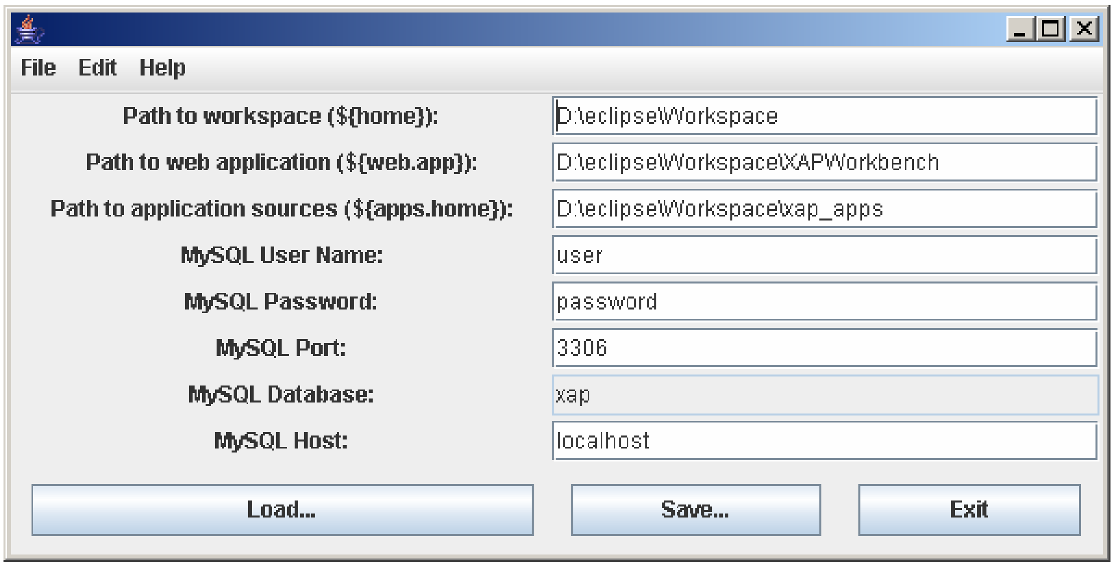


User Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

Category: Images

Applications:

Last we run configuratorGUI. We will choose the path to workspace and MySQL host.



Path to workspace (${home}):	D:\eclipse\Workspace
Path to web application (${web.app}):	D:\eclipse\Workspace\XAPWorkbench
Path to application sources (${apps.home}):	D:\eclipse\Workspace\xap_apps
MySQL User Name:	user
MySQL Password:	password
MySQL Port:	3306
MySQL Database:	xap
MySQL Host:	localhost

6. Discussion

In this chapter, we want to talk about some difficulties we met and some other functionality could be improved in the future.

6.1. Conclusion

To integrate several applications into a framework, we need to analyze the characteristics of all these applications. It is very important and necessary to understand the applications and get a main idea about what characteristics these application have in common. We also should know the relationship between these characteristics and the framework which we are going to create. After that, we should figure out how to integrate applications which have different characteristics into framework.

For example, in our project the three tools divide into two types---SWT and SWING. Our framework is SWT. So we focus on how to integrate SWING into SWT framework. Comparing several methods, we call SWING outside.

Our framework is extensible. If a user wants to add some other tools into this workbench, it is easy for him to do it. If the tool is SWT, he can call the tool inside the workbench and display it as part of the workbench. If the tool is not SWT, he can call it outside the workbench. He does not need to make a lot changes to the workbench.

To run tools is also very easy. After you create a project or open a project, choose an item from menu will make corresponding tool run either inside or outside.

6.2. Future work

There are some bugs existing, the future work first is to fix these bugs.

Now these three tools can run at the same time, maybe it is better to let one tool run at the same time.

And considering the Eclipse tool, we could see that the plug-in for Eclipse is really easy to install in it, just copy the .jar file into the specific directory, and import it. It is really great to use the plug-in tool. This could be the new functionality for this project in the future.

7. Reference

1. Paul F. Zarrella. “CASE Tool Integration and Standardization”, December 1990

http://www.sei.cmu.edu/publications/documents/90.reports/90.tr.014.html (2006.03.20)

2. Dewayne E. Perry, Gail E. Kaiser. “Models of Software Development Environments”

http://www.ece.utexas.edu/~perry/education/360F/tse17-3.pdf (2006.03.28)

3. Alfonso Fuggetta. “A Classification of CASE Technology”, December 1993

http://ieeexplore.ieee.org/iel1/2/6338/00247645.pdf?tp=&arnumber=247645&isnumber=6338 (2006.04.05)

4. Gabor Karsai, Andras Lang, sandeep Neema. ”Tool Integration Patterns”

http://www.isis.vanderbilt.edu/publications/archive/Karsai_G_9_0_2003_Tool_Integ.pdf (2006.04.05)

5. Rene Freude. “Tool integration with consistency relations and their visualisation”

http://www.es.tu-darmstadt.de/download/publications/koenigs/tool_integration_with_consistency_relations_and_their_visualisation.pdf (2006.04.20)