The Trinidad Platform

"Master Thesis"

Amsterdam, August 15 2006.

Ing. J. Jong
Universiteit van Amsterdam
Capgemini Nederland B.V.

Amsterdam, August 15 2006.

Author

Ing. J.J. Jong

0555177

Institute

One Year Master Course Software Engineering

Universiteit van Amsterdam,
Hogeschool van Amsterdam
en de Vrije Universiteit

Thesis Supervisor: drs. H. Dekkers

Company

Capgemini Nederland B.V.

Utrecht

Internship Supervisor: S. Hoogendoorn

Availability

Public domain

Preface

Before u lays the master thesis that covers the internship on the Trinidad Platform, carried out for Capgemini Nederland B.V. This internship is a conclusion to the Master Software Engineering taught at the University of Amsterdam (UVA).

Even though writing this thesis is an individual task, guidance and support do influence the final result. I would like to thank Capgemini for making this internship possible. In particular I would like to thank my internship supervisor, dhr. S. Hoogendoorn, for his guidance and support during the project. My thanks also goes out to all the members of Capgemini that participated in the interviews and sessions described in this work.

I would also like to thank the teachers of the University of Amsterdam, in particular my thesis supervisor, dhr. H. Dekkers, for his feedback and the time and effort taken in order to help me make this thesis to what it is now.

Jermaine Jong
Amsterdam, august 2006.

Contents

Summary

To stay ahead of the competition, Capgemini is standardizing the way software is developed which manifests itself in the realization of the Trinidad Platform. The Trinidad Platform is a fairly new method making its way through and being accepted by the organization. The Trinidad Platform executes on the vision that projects need to be empowered and highly standardized in order to achieve high productivity and high quality at the same time. The Trinidad Platform consists of a number of integrated core elements which are: an Agile methodology, the Trinidad Reference Architecture, Model Driven Development and a Community.

The Trinidad Reference Architecture needs to be documented. This document will serve as a reference for the Trinidad Reference Architecture and the Havana Framework. Just like the Trinidad Platform the document will be a dynamic one which will be updated and changed during the development of the platform. The intended audience for the document will exist of developers and architects wanting to get more insight in the technical details of the Trinidad Reference architecture.

Before documenting research has been done about the stakeholders, the criteria and the goal of the document. In order to give a structured reflection of the Trinidad Reference Architecture a clear way of documentation has been researched. The main sources of information during this process have been developers with experience of the Trinidad Platform. The information was gathered during architectural sessions which are more elaborately described in this document. During these sessions the most important components of the reference architecture were selected.

As an evaluation to the research the architectural overview of the Trinidad Reference Architecture has been assessed by members of the target audience. To get “objective” opinions people were chosen that were not directly involved with the research. The assessment was carried out with the use of a questionnaire. The process resulted in the architectural overview of the Trinidad Reference Architecture which can be found in Appendix A.

The Trinidad Platform will be developed and released to customers according to an open source model. This means that customers can use the platform for their own software development projects and at the same time contribute to further development and improvement of the platform. In order to successfully manage this process within a company like Capgemini a clear design is needed.

Research has been done in order to issue grounded advice about an open source model and version- and configuration management. The meanings of open source and general open source characteristics have been studied. In order to find out what makes an open source project successful, two successful open source projects have been studied. The way they managed the open source characteristics and their success factors have been analyzed. This research resulted in the formulation of the following hypothesis:

“If an open source project manages all the characteristics in a way that is appropriate for that specific project and that project takes the success factors into consideration the process will result in a successful open source project.”

As an evaluation to the research and a test of the hypothesis two failed open source projects have been studied. This evaluation has resulted in a final list with open source success factors.

The thesis ends with the strategy for an open source Trinidad Platform. The results of the internship, apart from this thesis itself, can be found in Appendix A: Trinidad Reference Architecture and Appendix B: Trinidad Open Source Strategy.

1 Project relevance

To stay ahead of the competition, Capgemini is standardizing the way software is developed which manifests itself in the realization of the Trinidad Platform. The Trinidad Platform is a fairly new method making its way through and being accepted by the organization. The Trinidad Platform will be released to customers according to an open source model. To do this, Capgemini needs insight in the pro and cons and the various ways to organize an open source community within a profit company.

Currently most of the knowledge about the Trinidad Platform is contained by a small group within the organization. In order to allow an easy spread of the platform, within Capgemini and throughout the community, clear (technical) documentation is mandatory. Therefore a detailed description of the most important concepts of the platform will be created during this internship.

1.1 The Trinidad Platform

The number of challenges modern software development projects encounter is expanding quickly. Think about integration with existing back-ends and third parties, service orientation, new platforms and media, and emerging technologies. For projects, executing on-time and on-budget becomes a diligent quest, with this ever increasing complexity. In order to deal with these challenges, Capgemini introduced the Trinidad Platform.

The Trinidad Platform executes on the vision that projects need to be empowered and highly standardized in order to achieve high productivity and high quality at the same time. The Trinidad Platform consists of a number of integrated core elements.

- *Agile methodology*
  A flexible agile methodology, using best practices from Rational Unified Process (RUP), MSF Agile, extreme programming and Smart. An agile methodology is used to guarantee frequent, high quality delivery of working software. With this methodology comes a clear and easy-to-use estimation technique, based on pragmatic use cases, modeling guidelines and an online use case based planning and measurement tool. Furthermore, additional project and process support can be achieved by executing the customer's project in one of Capgemini's Accelerated Delivery Centers (ADC).

- *Trinidad Reference Architecture*
  Projects executed with the Trinidad Platform de facto apply platform independent multi-tier reference software architecture. Be it web, mobile or Windows development, employing databases, service oriented architectures (SOA) and enterprise busses.

  The reference architecture is supported by a broad and extensible framework, called the Havana Framework. This allows for maximum re-use of functional, technical built-in and third party services and components, such as authorization, SharePoint, web services, BizTalk, Microsoft Dynamics, and SAP. Capgemini gathered years of experience in constructing and using frameworks, such as Sculptor, CSLA and numerous project specific workbenches.

- *Model Driven Development*
  High quality code is delivered at high speed. This is facilitated using model driven architecture and development (MDA). The Tobago MDA Generator uses UML models, and produces code according to patterns defined. Model driven development guarantees high quality design and code and keeps testing effort low.

- *Community*
  Capgemini is keen on collaborating with its customers on the Trinidad Platform. Thus, an open

source community-style business model is applied, were customers worldwide can use and contribute to the platform. In this way, the knowledge and experience of the platform expand at high pace.

- *People*
  The single most important asset in software development projects is people. In the Trinidad Platform, training and coaching are key towards success. Workshops are therefore available for each of the core elements of the platform, including estimation techniques, modeling, and using code generation and frameworks.

The Trinidad Lifecycle

The agile methodology used within Trinidad projects is called the Trinidad Lifecycle. The Trinidad Lifecycle is the beating hart of every Trinidad project.

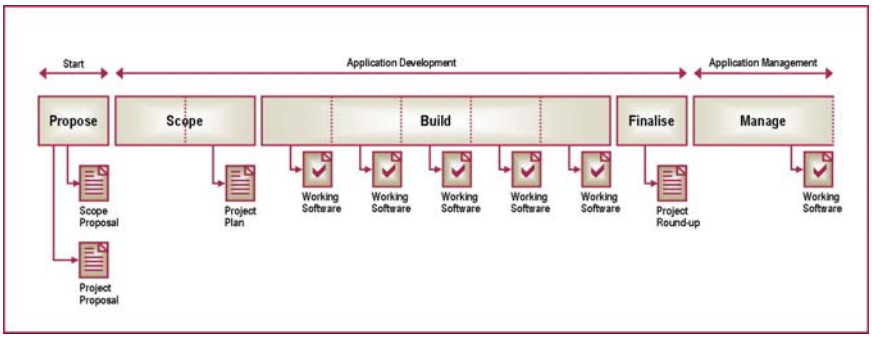

Fig. 1.1 Trinidad Lifecycle

The Trinidad Lifecycle exists of the following phases.

- *Propose*. The project's scope, size and complexity are determined roughly during a number of short intensive workshops. This stage leads to the initial project proposal.

- *Scope*. The project proposal is elaborated upon, again in workshops, leading to the plan of approach for the project. This stage includes smart use-case modeling and domain modeling, stakeholder and risk analysis, resourcing, and an estimate for the project.

- *Build*. The "Build" phase stands in light of interactively realizing software. "Build" is divided into short iterations of preferably two weeks. During these iterations a number of smart use cases are realized. At the start of an iteration the use cases that will de realized are selected. This happens in the sub-phase "Plan". Next every iteration has a sub-phase "Build", in which the use cases are elaborated and realized. Deployment happens in the sub-phase "Run".

- *Finalize*. During the *Finalize* phase the project is finalized and evaluated.

- *Manage*. The (ongoing) stage *Manage* executes the maintenance of the delivered software. During this stage again smart use-cases underlie the possible changes in the software. The stage *Manage* is usually executed in monthly or two-monthly iterations.

1.2 Internship

The internship can be roughly divided into two parts. The first part of the internship is a description of the Trinidad Reference Architecture and the second part will be issuing an advice about an open source strategy for the Trinidad Platform. These two parts of the internship will be described in the following chapters of this document.

1.3 Project approach

In order to successfully finish this project a certain approach is followed. The approach can be divided into 3 sections; there was the initial prework and preparation before starting the project, following by the approach to documenting the Trinidad reference architecture and at last the approach to setting up a Trinidad open source model. The most important steps are shown below in chronological order.

1.3.1 Prework

Internship description
Prior to the internship a clear description of the assignment was formulated and described, along with the project's scope. This document was reviewed by the organization to make sure both parties were on the same line.

Literature study
The execution of the assignment starts with a literature study. Within this study literature is gathered that can be helpful during the project. As project continues the list of literature eventually expanded with additional literature.

1.3.2 Documenting the reference architecture

The process of documenting the Trinidad Reference Architecture is described in chapter 2. In order to keep a clear structure in this document the approach to the research and documentation of the reference architecture are bundled together with the description of this process. Chapter 2.2 gives a detailed overview of the steps taken in the approach.

1.3.3 Trinidad Open Source

The approach taken in the research on the Trinidad open source advisory model is also described with the description of this process, which can be found back in chapter 3. The reason for this is also to keep a clear structure in the document.

1.4 Project risks

Like every other project there are risks that need to be taken into account during the project. Below are some of the risks the project carries and how these risks will be dealt with during the course of the project.

- The Trinidad Reference Architecture is very extensive, therefore it takes a lot of effort to fully understand and document it. However a clear understanding of the reference architecture is mandatory for performing research on a suitable open source model. Because of the limited amount of time available the process will have a tight schedule and the description of the reference architecture will start right after the literature study.

- The internship contains a lot of dependencies. Most of the knowledge regarding the Trinidad Platform has to come from more experienced Trinidad developers within Capgemini. These

developers have a tight schedule so clear appointments need to be made way in advance in order to acquire the needed information.

1.5 Knowledge

To successfully complete this project a certain degree of knowledge is required. Qualities which the performer must possess have to be among other things knowledge of/experience with the reference architecture of the Trinidad Platform, UML, use cases, design patterns and the open source model. This knowledge will come from different sources, varying from literature to people.

1.6 Deliverables

The project knows the following deliverables:

- Research report (thesis) of the performed research
- Description of the Trinidad Reference Architecture
- Trinidad open source advisory report

2 Trinidad Reference Architecture

The focus of the first part of the project is documenting the Trinidad Reference Architecture. The reference architecture is a very important part of the Trinidad Platform. The reference architecture, supported by the Havana Framework, builds upon knowledge gathered over the last years. It takes time to uncover and fully understand all the possibilities of the reference architecture. This chapter covers the actual research that has been done to find out how the Trinidad Reference Architecture works and document it.

2.1 Research questions and goals

The Trinidad Reference Architecture needs to be documented. This document will serve as a reference for the Trinidad Reference Architecture and the Havana Framework. Just like the Trinidad Platform the document will be a dynamic one which will be updated and changed during the development of the platform. The intended audience for the document will exist of developers and architects wanting to get more insight in the technical details of the Trinidad Reference architecture.

A thorough research of the reference architecture is mandatory in order to describe it in a clear and correct way. The research is based on the following main questions:

- *What are the criteria for the reference architecture description?*
- *How can the reference architecture best be documented in a clear way?*
- *What are the most important components that form the reference architecture?*
- *Which design patterns are contained by the reference architecture and how are they used?*

2.2 Approach

In order to structure the research an approach is needed to document the Trinidad Reference Architecture. Structured and well over thought research promotes the results off this process. The approach exists of the following steps:

Goal of the document
The goal of the document is one of the most important issues to deal with from the start of the project. The goal of the document determines the stakeholders, the criteria, the way of documenting and even the evaluation. The first step in the research is getting a clear picture of the goal of the architectural overview.

Analyzing Stakeholders
The goal of the document automatically points out the stakeholders involved. A stakeholder analysis will be performed to find out which concerns the stakeholder have. The document will be created according to this information in order to meet these concerns.

Defining criteria
After the goal of the document and the concerns of the stakeholders are known, criteria for the document can be formulated. These criteria will be taken into account and will be leading during the creation of the reference architecture document.

Documenting
There are many ways to describe a reference architecture. Before documenting it is wise to decide on a suitable documentation manner that will allow the goal of the document to be met. This step covers the research and decisions that have to be made in order to decide on a suitable structure for the document.

Identifying important components and design patterns
The Trinidad Reference Architecture exists of a lot of components and design patterns. The most important ones need to be identified so that they can be described. This step covers the method used to identify the most important components and design patterns within the Trinidad Reference Architecture.

Evaluation
The evaluation is a very important step in the research. How do you prove that the document is correct, meets the criteria and is useful to the organization? In order to do this the following activities will be carried out:

- Matching the document to the criteria that were defined in the beginning of the project.
- Assessment of the document by the target audience, which in this case are developers and architects. The target audience can give the best feedback about the extent to which the goal of the document is reached.
- Feedback (interim) from the project managers of Capgemini.

2.3 Prework

After the global introduction to the Trinidad Platform, the reference architecture was studied in more detail. The first step was to find out what is understood by a reference architecture. In [1] the following definition of a reference architecture is given:

"A reference architecture is, in essence, a predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed, and proven for use in particular business and technical contexts, together with supporting artifacts to enable their use. Often, these artifacts are harvested from previous projects."

This definition clearly describes what a reference architecture is because it points out its essential characteristics, like a predefined pattern or set of patterns, supported by artifacts harvested from previous projects, etc. Therefore this definition was used during the project. The Trinidad Reference Architecture meets all the criteria stated in the definition above.

2.4 Goal, stakeholders and criteria

The goal of the architectural overview is to serve as a reference for developers and architects who want to gain more insight in the technical details of the Trinidad Reference Architecture and the Havana Framework. A stakeholder analysis has been done to establish what the different stakeholders required from the description. The two main groups of stakeholders are developers and architects. Below are the most important concerns these two groups have:

Developers
Developers that are new to the Trinidad Platform and development with the Havana Framework will use the document. By reading the document developers can get an overview of all the important technical aspects within the framework and the way these aspects link to each other. The document also shows the rationale behind the technical aspects. The reason, benefits and even the possible alternatives, if any, are documented. Furthermore experienced developers will use the document as a reference during development.

Architects
Architects new to the Trinidad Platform can use the architectural overview in order to get a clear picture of

the reference architecture and the different variations that are possible within the platform. Architects will also use the document as a reference.

Taking the concerns of these stakeholders into consideration the following main criteria were formulated:

- Clear scope, The Trinidad Reference Architecture is part of the Trinidad Platform which contains a lot of elements that speed up development. A lot of other steps are involved (like use case modeling and code generation) before using the Trinidad Reference Architecture. However the focus of the architectural overview needs to be only on the reference architecture.

- Reference for architects and developers, the architectural overview needs to serve as a reference for architects and developers new to the Trinidad Platform. The architectural overview needs to contain a clear description of the technical aspects of the reference architecture (design patterns, etc.)

In order to reach these criteria the first step was to find a logical way of organizing the information in a document.

2.5 Documenting the reference architecture

Both the stakeholders have different initial concerns. The Trinidad Reference Architecture starts out with a roadmap which marks the chapters that are most important for a particular stakeholder. However, all the chapters in the architectural overview can be useful to both stakeholders; this roadmap only provides an aid for the reader. The chapters below show the structure of the document.

2.5.1 Platform Overview

The Trinidad Reference Architecture is part of the Trinidad Platform. In order to give the reader an overview of the role the reference architecture plays within the platform, the architectural overview starts out with a concise description of the platform itself and every component contained by it.

2.5.2 Architectural Layers

The Reference Architecture exists of a couple of layers. In literature there are a number of models that can be used to describe an architecture. Initially the 4+1 view model as described in [2] was chosen to document the reference architecture. This model can show a software architecture from multiple stakeholder views and different levels of composition. The model contains 4 views which, combined with scenarios, result in a well documented software architecture. The 4+1 view model implies using different views for different stakeholders while describing the architecture. However, in a reference architecture the amount of detailed information is limited, therefore making it hard to create all of the different views in the model. Because of the lack of information this would result in poorly documented views that wouldn't contribute anything to the document. A reference architecture only contains properties that are common to every project that is based upon that reference architecture. Once a project is launched the other views can be build for that specific project. Below is an overview of the views in the 4+1 view model and the information that is required in order to create these views.

Logical view
Shows the object model of the design and contains class diagrams. It isn't possible to create a class diagram for the Reference Architecture because the classes in the application depend on the specific project based upon the reference architecture. A class diagram of all the base classes could be created instead, but this wouldn't contribute much to the reader.

Process view
Captures the concurrency and synchronization aspects of the design. This view describes the tasks of the application and the communication between these tasks. The flow of the program depends on the kind of

application being built with the reference architecture. This makes this view impossible to develop on forehand.

Physical view
Mapping of software onto the hardware. The Trinidad Reference architecture isn't bound to specific hardware. Trinidad applications can operate on any kind of hardware. Therefore choosing a much used hardware setup, like that of a webapplication, and creating the Physical view wouldn't be useful because of the many variations that can be present in hardware setups. The view would just represent a single situation.

Development view
Static organization of the software in its development environment. This view describes development issues like the layers of the application. The layers of an application are just the specific issues that are captured within a reference architecture. This view can be used in order to describe the Trinidad Reference Architecture.

The Trinidad Reference Architecture is a reference for the development of applications, so the stakeholders are interested in development issues. Of all the views in the 4+1 view model the development view is the one that can be created fully, based upon the information available. This view also captures all the information required by the stakeholders. Therefore the development view can be used to describe the reference architecture.

The complete 4+1 view model is not suitable for documenting the Trinidad Reference architecture. The three other views, besides the development view, can only be created in such a limited way. This wouldn't contribute anything to the document, other than confuse the reader. Therefore the decision has been made to use only the development view.

For documenting the Trinidad Reference Architecture the development view is used. This view shows the layered organization of a standard Trinidad application based upon the reference architecture. This development view is enhanced with the objects that characterize every layer (figure 2.1). The functionality and purpose of every layer is described in the architectural overview.

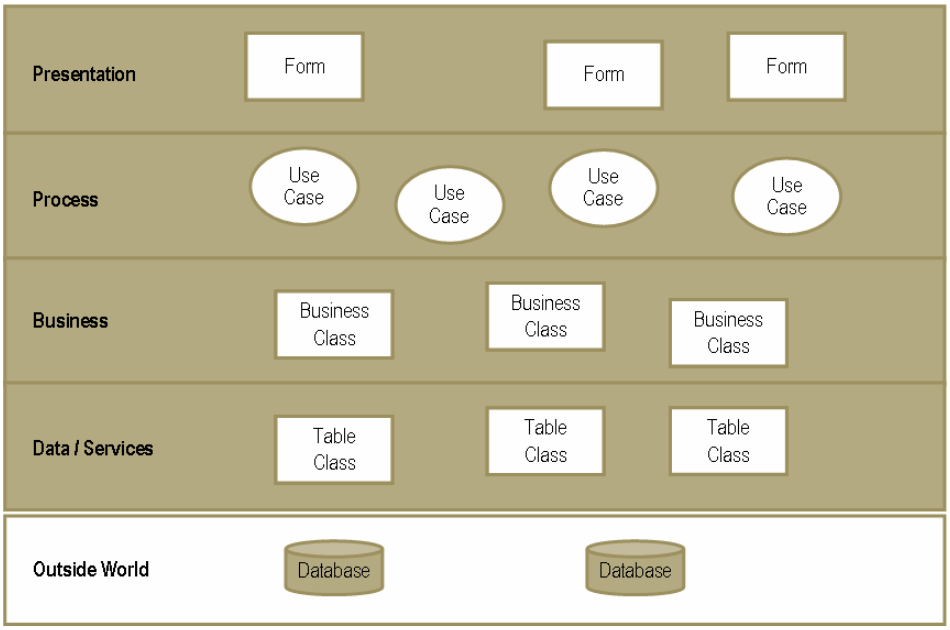

Fig. 2.1 Reference Architecture

2.5.3 Variations

There are a number of variations possible with Trinidad based applications, which are not shown in this view but clearly described in the architectural overview (appendix A.) Aimed at Service Oriented Architectures a Trinidad based application can for example be created as a service providing or service consuming application.

In order to clearly document these variations research was done about the applicability of the product line notation as described in [3]. There is a difference between a product line architecture and a reference architecture. A product line architecture is used to create a family of products that mostly contain commonalities and a few variation points. A reference architecture on the other hand is used to create all kinds of applications. The focus of a product line lies in the commonality and variations between features, while applications based on the reference architecture might have no commonality whatsoever.

However the concept of variation points used to describe the variations between the members in a product line can be used to describe the different variations of the Trinidad reference architecture. After identifying the variation points they were each described in a different picture. The product line notation prescribes the creation of a single architectural image which shows all the possible variations, sometimes completed by a feature tree. However, the variation points in the Trinidad reference architecture are all technical. Displaying each in a separate picture makes it possible to show the varying components along with the layers they are active in. This makes the variations easier to understand. Placing these variations in a single picture could decrease the readability of the picture. Therefore the product line notation was rejected.

A feature tree could have been used to emphasize the variations in writing and drawing. However the separate architectural pictures already give sufficient overview and clarity that a feature tree was abundant and therefore omitted. The results can be found back in the architectural overview.

2.5.4 Technical components and design patterns

The technical components in the document are organized per layer. From every layer the most important technical aspects and their main functionalities are described.

The technical components and design patterns within the Trinidad Reference architecture needed to be described in a structured and concise way. A clear way to describe these technical components is by using a pattern. A pattern gives the user an unambiguous outline of every pattern, and the user knows exactly where certain information is described. Furthermore many books on design patterns for example describe them according to a pattern, leading to clear, easy to read books.

The “Gang of Four Template”, which is used to describe the design patterns in [4] (and can be found at [5]), is a clear and suitable pattern to use in the architectural overview. In order to make the pattern suitable not only for design patterns, but also for the other technical components within the Trinidad Reference architecture the pattern was adapted and some of the sections left out (fig 2.2).

Intent
Short statement that answers the following questions: What does the component do? What is its rationale and intent? What particular design issue or problem does it address?

How (Motivation)
A scenario or example that illustrates a design problem and how the component can solve the problem. The scenario will help you understand the more abstract description of the component that follows.

Benefits
What are the benefits from using this component.

Consequences
How does the component support its objectives? What are the trade-offs and results of using the component? What aspect of system structure does it let you vary independently?

Alternatives
What are (if any) alternative ways to solve this particular design issue or problem?

Fig. 2.2 Technical component template

2.6 Identifying important components

Having decided on the structure used to document the reference architecture, it became important to determine what to describe in the architectural overview.
"*What are the most important components that form the reference architecture?*"

In order to answer this question "architectural sessions" were held with developers of the platform. In an architectural session a few developers came together in order to explain the components of the reference architecture. The goal of such an architectural session was to gather information about the framework and the components contained by it. These sessions varied from 1 to 1,5 hour. Mostly the components were handled per layer.

Researching and examining the most important components within the reference architecture took a great deal of time. There are a lot of technical components within the reference architecture which all contain rationale, variations and sometimes a number of alternatives that could have been chosen. All these aspects needed to be understood in order to describe them. A lot of components within the reference architecture are based upon well-known design patterns. Literature from Martin Fowler [6] and "The Gang of Four" [4] were used in order to identify, understand and describe these components.

The components within the framework are linked to each other instead of being individual pieces. The inter-layer components were first examined and described to get an overview of the total structure and the relations between the layers. This made it easier to describe the layer-specific components because the total picture was somewhat clear. However the description of every layer contributed to the total image of the flow within the reference architecture.

2.7 Evaluation

In order to assess the correctness of the Trinidad architectural overview it was evaluated after documentation. This evaluation was done in the form of an assessment, which exists of testing the document against the formulated criteria and the initial goals. The research done in order to create the architectural overview also been evaluated.

2.7.1 Assessment

The Trinidad architectural overview was assessed by members of the target audience. To get "objective" opinions people were chosen that were not directly involved with the research. A developer and a member of management were chosen to asses the document. The assessment was carried out with the use of a questionnaire.

The document was assessed by a developer new to the Trinidad Platform. Because the developer doesn't have any knowledge of the platform the outcome can say something about the extent to which the architectural overview helped get a clear overview about the Trinidad Reference Architecture. The document was also assessed by someone active in management of the Trinidad Platform, but also has thorough knowledge of the platform and the reference architecture. This assessment was aimed more at the applicability of the document. The combined results of the assessment are summarized below, divided into positive and negative comments:

Positive

- *Clear overall impression;* The document gives a clear overall impression of the Trinidad Reference Architecture. Even for developers that are new to the platform the document is understandable and instructive.
- *Correctness;* The technical aspects in the document are described in a clear way, with a correct description of their specifics. The pattern for describing these technical aspects aided the readability of the document.
- *Most important concepts;* The architectural overview contains a description of the most important and interesting technical aspects of the Trinidad Reference Architecture. All aspects that speed up development are documented.
- *Scope;* The document has a clear scope. The Trinidad Reference Architecture is the only part of the Trinidad Platform that has been fully described.

Negative

- *Starting a Trinidad project;* The developer new to the Trinidad Platform hasn't been able to participate in a Trinidad project based upon the information in the architectural overview.
- *Applicability;* The document isn't mature enough to be used by the customers of Capgemini using Trinidad through the open source model. There is still information missing in the document for it to meet the concerns of the customers. However, the document forms a good basis.
- *Class Overview;* At the current version the Trinidad Reference Architecture lacks an overview of the base classes that are to the disposal of the developers. This might be helpful for the developers reading the document because it provides an overview of the whole.

2.7.2 Research questions

From the results of the evaluation the following conclusions concerning the research questions can be drawn.

How can the reference architecture best be documented in a clear way?
Research has been done in order to come up with a clear way of documenting the Trinidad Reference architecture. During the assessment the readers found the document clear and easy to read. These factors provide enough ground to state that this research question has been answered during the research.

What are the criteria for the reference architecture description?
During the evaluation the criteria haven't been assessed enough to conclude if this research question has been answered. However the results of the evaluation state that the architectural overview is not directly applicable to customers of Capgemini, even though the formulated criteria didn't require this. This indicates that the initial criteria may have been incorrect or that the criteria experienced a change during the second part of the research.

What are the most important components that form the reference architecture?
During the research "architectural sessions" were held in order to elicitate the most important components of the reference architecture. While documenting, the architectural overview has been iteratively assessed by the internship supervisor and other developers well known with the Trinidad Platform. The evaluation of the architectural overview, partly done by someone from the target audience with extended knowledge of the platform, also results that the most important components of the reference architecture have been documented.

Which design patterns are contained by the reference architecture and how are they used?
There are a lot of design patterns used in the Trinidad Reference Architecture. Not all are yet described in the architectural overview. The choice has been made to focus on the most important components, and describe the design patterns that are present in them. The remaining design patterns can be documented in future versions of the document.

2.7.3 Research

Research has been done in order to document the right things about the Trinidad Reference Architecture in the right way. A detailed approach to the project has been thought out on forehand. Figure 2.3 gives a graphical overview of the research process as it has been carried out within this project.

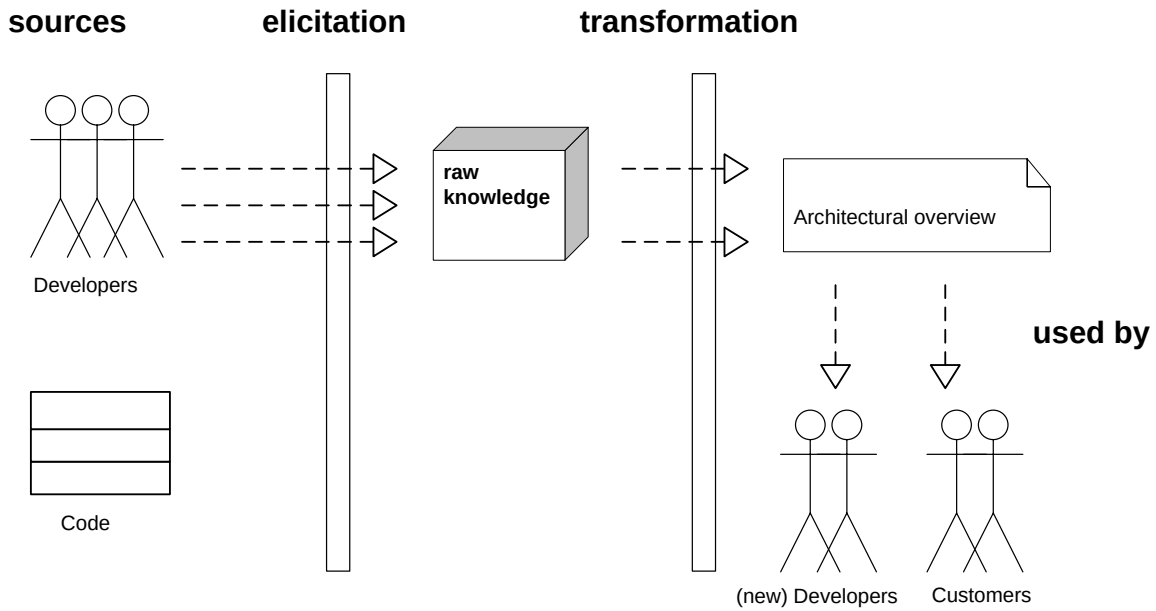

Fig. 2.3 Research process

Sources
The input for the architectural overview came from the developers. It was an option to use the code as input for the document, either with or without the developers. However exploring the code without any knowledge of the reference architecture would have been hard. Furthermore the developers had such good knowledge of the Trinidad Platform that they were trusted as sole source for the document.

Elicitation
The elicitation of the information went smooth. If the research had to be done over, the elicitation process could have been structured more, in order to perfect it (think about standard elicitation forms, standard session setup, etc.).

Transformation
The transformation of raw knowledge to the architectural overview, the process of documentation could have been improved. Because of the complexity and size of the Trinidad Platform the process of documenting the platform took far more time than the research on it. A different approach tot the project would have been one where the process of documenting itself got more attention. Below are a few ways in which this could have been done.

- Using a persona and scenarios; Persona's and scenarios could have been created in order to draw out and represent the use of the document by the target audience. The document would have been written accordingly.
- Intermediate assessments; The use of assessments not only at the end of the research but also interim can reveal any shortcomings of the document early.
- Target audience involvement; More involvement and interaction with the target audience during the creation of the document. Without any knowledge of the platform the author could have even stepped in the role of a new Trinidad developer, in order to discover the information required in the document.

In fact all the possibilities named above all relate to the following; better research on the way the document will be used.

Future work on the architectural overview of the Trinidad Reference Architecture would be extending it with all of the technical aspects instead of just the most important ones. This process of documenting resulted in the architectural overview of the Trinidad Reference Architecture which can be found in Appendix A.

3 The open source model

The open source principle is becoming more and more popular to the developers of software. The Trinidad Platform will be developed and released to customers according to an open source model. This means that customers will use the platform for their own software development projects and at the same time contribute to further development and improvement of the platform. In order to successfully manage this process a clear open source strategy including version- and configuration management is needed. All parties need to be able to easily present and integrate their contributions to the architecture and the framework.

There are benefits but also disadvantages attached to releasing software under an open source model. In order to set up a grounded advice for the Trinidad open source model research about the characteristics of open source is necessary.

3.1 Approach

Because open source is a topic that is becoming more popular, the available information about open source is growing. In order to make a way through this pile and find the required information structured research is necessary. The following approach has been used during this research:

Open source
As a starting point of the research the question “What is open source?” needs to be answered. Furthermore research will be done on the reason open source projects is interesting for companies, and why an open source Trinidad Platform is interesting for Capgemini.

Characteristics
In the extent of what open source is, the characteristics of open source projects and the tools required to launch an open source project will be analyzed. Tools can differ from documentation and support to configuration management and development tools. This list of characteristics will serve as input for the Trinidad open source advisory report.

Successful open source projects
Two successful open source projects will be analyzed. They will be compared with the list of open source characteristics in order to see how they managed them. Research will be done in order to try and reveal the succes factors of these projects, or maybe even come to the conclusion that some of the open source characteristics are success factors themselves.

Expert interaction
Besides the two successful open source projects input of experts on the area of open source will also be used in order to complete the success factors and characteristics of open source projects.

Hypothesis
Armed with the characteristics and succes factors of open source projects, at this point in the research a hypothesis can be formulated.

Proof of concept
To proof the hypothesis wrong or right it needs to be tested. The proof of concept will be done in the following ways:

- Comparing the succes criteria to failed open source projects. Research will be done to see if the failed project lacked essential success criteria. The failure factors of the open source projects can

say a great deal about the list of success criteria. The analysis of these projects may even result in some fail criteria, which can be used in the final advice.

- Review of the advice by experts in the area of open source.

Advisory report
After the research is done the results will be documented in an advisory report, intended for Capgemini.

3.2 What is open source

Open source software automatically makes people think of "free" software. Actually open source software is much more than that. In [7] the author gives the following definition of open source:

"Open source software is any computer software distributed under a license which allows users to change or share the software's source code"

The open source Model allows users to develop and contribute to a project voluntarily, which eliminates the need of an actual supplier. The software is shared and can be used freely by anyone. Suppliers can choose to go open source with their products because of the rapid development of their projects that results from the many developers. Communities of developers and contributors are formed where users share knowledge and report bugs and feature requests. Open source doesn't just mean access to the source code. The distribution terms of an open source program must comply with a number of criteria. These criteria are recorded in the open source Definition. Bruce Perens gives in [8] a short summary of these criteria including a link to the full version, which can be found at http://www.opensource.org/docs/definition_plain.html.

There are a lot of different applications of open source methods, combining properties of markets and communities. In order to set up an open source model it is necessary to know exactly what the open source model is. Most open source models share the following characteristics [7]. This list of characteristics serves merely as an aid to helping the reader form the picture of an open source model. The research will not be based on this list, however most, if not all these characteristics are already "touched" by the defined research questions.

- *Transparency*, potential contributors need to understand what it is they're contributing to.
- *Examination of participants only after they've got involved*, reducing the barriers to involvement by allowing absolutely anyone to get involved.
- *Low cost and ease of engagement*, many people can get involved at no additional outlay beyond what they already spend on their computing.
- *A legal structure and enforcement mechanism*, open source does not mean a free for all. Instead it depends on a clearly defined legal framework which shapes the incentives for participation.
- *Leadership*, most open source projects have some centralized element of leadership or control that sets the general direction and ethos assigns tasks and acts as an editor, approving changes to the source code.
- *Common standards*, Successful open source projects rely on open, free-to-use standards, and they create new, open, free-to-use standards for their users.
- *Peer review and feedback loops*, the open source collaborative approach manages to produce high quality work because of the many reviews people among themselves.
- *A shared conception of goals*, open source projects may deal with internal dissent about particular choices and directions but there is enough of a common conception of the good to make each project thrive.
- *Incrementalist*, small players can still make useful contributions.
- *Powerful non-monetary incentives*, open source methods contain the ability to replace traditional cash incentives with non-monetary ones like motives of social or personal fulfillment.

3.3 Capgemini's vision on open source

The Trinidad Platform is a new development platform within Capgemini. In the short time that the platform exists it has seen a growth of acceptance and interest from other parts of the organization. There are a few projects already carried out on the Trinidad Platform.

Open source projects are booming, even though open source seems like a counterintuitive strategy. Some of the open source supporters even claim that open source projects can withstand Brooke's Law [9] which sounds as follows "Adding manpower to a late software project makes it even later". In an open source project people continuously join or quit development. Even though this claim is not completely true, because most open source projects don't have a schedule so the words "late" and "later" used in Brooks Law are of no meaning here, open source projects are characterized by rapid development by a great number of developers.

Custom software development projects are diminishing. Companies tend to choose offshoring or standard packages, which don't fully fit their needs over custom software development. The main reason for this choice is a financial consideration. In order to compete, custom software development needs to offer benefits over these new development ways. Capgemini believes in the power of the Trinidad Platform and the many benefits it contains for custom software development. Trinidad stands for cheaper, faster, tailor fit custom software development.

The Trinidad platform can be interesting for a lot of custom software development projects. The aim of Capgemini is to spread the Trinidad Platform throughout the world. Capgemini wishes to offer the Trinidad platform to customers in order to let them carry out their own software development projects. If we look at the above, this would fit perfectly within an open source model.

As the founding company behind the Trinidad Platform the Capgemini's name will be spread along with the platform. For support participating companies will turn to Capgemini which has good economical side effects. Capgemini will train and school developers on the Trinidad Platform and deliver support. In exchange, the users of the platform need to contribute any modifications or additions they've made to the platform.

Implementing an open source model has benefits, but doesn't come without risks either. The risks and the benefits Capgemini expects of an open source model are described below.

3.3.1 Technical benefits for Capgemini

There are a number of expected economical benefits for Capgemini from an open source Trinidad Platform. These benefits depend on the chosen business model and open source license. Besides economical benefits Capgemini expects and wishes to achieve the following technical benefits with the release of an open source Trinidad platform:

- Rapid development of extensions to the Havana Framework. The core of the Havana Framework is mature and stable enough, so at the right moment in time to be released as an open source project.
- Quick bug report and fixes, patches, ports to new platforms, etc.
- Growth and evolution of the Trinidad platform and the community.

With the Trinidad platform Capgemini wants to lean more and more to standardization, because standardization is the key to rapid custom software development. The platform is relatively young and still has a lot of work. However Capgemini knows from experience that most additions, like a specific logging mechanism, are only build when there's a customer who actually has the need for it. Releasing the platform as open source will speed up the development of such extensions rapidly.

3.3.2 Technical risks for Capgemini

The risks for a project to go open source are just as important as the benefits. Before starting an open source project many companies have the same worries about potential risks that such a project may carry [10]. Capgemini oversees the following main risks when starting an open source Trinidad platform:

- Unstructured development process, a lot of open-source software development is still done ad hoc, without good development practice. Most developers start coding before writing down a design [11]. This slows down open source development, while programming is likely to be more regulated and professionalized.

- Fragmentation of the products into incompatible versions, if everybody can issue releases and bring out new versions there can be a spread of different incompatible versions. There need to be a controlling entity in order to structure this.

- Exposing strategies to competitors, the open source model implies sharing strategies and methodologies. This can be a risk, but with good initial support from Capgemini, competitors will rather join development and require support than start an own platform.

- Becoming a product vendor when that is not Capgemini's primary business.

3.4 Research question and goals

There are a lot of concerns that need to be taken into account before setting up an open source model within a company like Capgemini. The goal of this research is to issue a grounded advice about an open source model and version- and configuration management. The advice will be based on thorough research, among others about literature regarding open source models and version and configuration management, license structures for open source projects, and research about the extent of support that Capgemini wants to deliver to the framework. The main research questions are:

- *What are the different license structures in open source projects and which one suits the needs of Capgemini best?*
- *What are the motives and benefits for developers, architects or other interested parties to participate and contribute to an open source Trinidad Platform?*
- *How do you safeguard the quality of code within an open source project?*
- *How do you implement an open source strategy, what are the tools required in order to launch a successful open source project? Tools can differ from documentation and support to configuration management and development tools.*
- *What are the requirements for a successful open source project?*

As a basis for the research two successful open source projects will be analyzed. The research questions can be compared to the two projects. The conclusions drawn from the way these projects are set up and managed will be used as input for creating the advisory report for the Trinidad Platform. The succes factors of these projects will also be examined. The two open source projects that will be compared are Hibernate and the Spring Framework. These are both architectural frameworks which support development, therefore sharing a lot of resemblance with the Havana Framework.

The Trinidad reference architecture description will, once the open source model is implemented, also function as a tool in the community, serving as a reference not only for employees of Capgemini, but to everyone using the platform.

3.5 Open Source Licenses

Open source projects are based upon the willingness of other developers to work for free. In order to get developers to contribute to a project (besides their personal motives which will be discussed in the next chapter) they need to be provided with some kind of freedom. Developers feel comfortable contributing to any open source project because they are assured of the following rights [8]:

- The right to make copies of the program, and distribute those copies
- The right to have access to the software's source code, a necessary preliminary before you can change it.
- The right to make improvements to the program.

These rights keep all the contributors at the same level relative to each other. These rights are formalized in the form of an open-source license. Open source software is very much copyrighted software, with the difference that the program's license grants the user rights that give much more freedom than conventional software. The Trinidad Platform will need a license that connects to the goals and vision of Capgemini.

Below are a few standard license agreements [8] that can be used for or modified to fit an open source project.

Public Domain License
Public domain stands for a program which has no copyright at all. Every one can do as he or she pleases with the program (including re-licensing it and / or removing it from the public domain).

The GNU General Public License (GPL)
This license prohibits developers from making changes to open source products and not contributing those changes back to the developer community. These modifications must also be distributed under the GPL license. GPL programs are not allowed to be mixed with non-free software.

The GNU Library General Public License (LGPL)
The LGPL license is a derivative of the GPL. Unlike the GPL, a LGPL licensed program can be mixed with non-free software.

X, BSD and Apache Licenses
These three licenses are relatives from each other. The main difference between these and the GPL license is that modifications can be taken private.

The Artistic License
The Artistic License is known for its ambiguous and self contradicting rules. Therefore programs using this license are often dual-licensed with the GPL license.

The Netscape Public License (NPL) and the Mozilla Public License (MPL)
The NPL license was created by Netscape when they made the Netscape Navigator and includes rules that give Netscape special privileges that only apply to them. In order to use this license in public Netscape created the MPL, which is a copy of the NPL but without the clause that grants Netscape special rights to the software.

There are many more open source licenses. The full text of the above and the other available open source licenses can be found at http://opensource.org/licenses/. A common characteristic of all open source licenses is that they each disclaim all warranties.

The table below shows the important characteristics of the mostly used open source licenses. This list can be of help when choosing an appropriate license for the Trinidad Platform.

License	Can be mixed with non-free software.	Modifications can be *taken private* and not returned to you.	Can be re-licensed by anyone	Contains special privileges for the original copyright holder over your modifications.
GPL	no	no	no	no
LGPL	yes	no	no	no
BSD	yes	yes	no	no
NPL	yes	yes	no	yes
MPL	yes	yes	no	no
Public Domain	yes	yes	yes	no

3.5.1 Business models

A frequently asked question is how to generate revenue from an open source project. The idea of giving away software eliminates the traditional revenue from license fees. In order to still make profit it is mandatory to select a suitable business model and execute it well. As Daly [12] states the answer doesn't lie within the product itself, but within the industry and business surrounding the product. A suitable business model will be selected for the Trinidad Platform. Hecker [10] identifies eight styles (shown below) from which one may be selected for the Trinidad Platform.

- *Support Sellers;* Revenue comes from media distribution, branding, training, consulting, custom development, and post-sales support.
- *Loss Leader;* A no-charge open-source product can be used to lead customers towards the company's own proprietary software.
- *Widget Frosting;* Used by hardware manufacturing companies that use the open source model to develop software like drivers and applications to go with the hardware.
- *Accessorizing;* A company distributes books, computer hardware, and other physical items associated with and supportive of open-source software.
- *Service Enabler;* Open source software is created and distributed primarily to support access to revenue-generating online services.
- *Brand Licensing;* One company charges other companies for the right to use its brand names and trademarks in creating derivative products.
- *Sell it, Free it;* A company's software products start out their product life cycle as traditional commercial products and are then continually converted to open-source products when appropriate.
- *Software Franchising;* This is a combination of the support selling and brand licensing model. A company may wish to sell trademark rights or franchise rights with materials and training to other businesses that wish to enter the same market space. In this way, revenues would come from franchise and trademark royalties.

3.6 Developers motives

Even though the open source strategy may seem counterintuitive as it goes against years of tried and true commercial practice [10], there are still a lot of developers voluntarily contributing to open source projects. The question that rises is what drives these developers to do this. After all developers are the most important factor in any open source project. Without developers willing to participate and contribute time and effort in an open source project it will surely fail. In order to set up a successful open source project it is necessary to understand these motives and use them to attract developers.

As the last open source characteristic from chapter 3.2 already states there are non-monetary motives involved. There have been many studies about the motivation of open source software developers. In [13] the authors make the distinction between individual psychological factors (internal factors) and rewards (external factors). Internal factors, also called intrinsic motivation [14], are certain activities and behaviors that people like to perform naturally. If we put this in the open source context, this category describes programmers who make a hobby out of programming and are motivated from the feeling of competence, satisfaction and fulfillment that arises from writing programs [13]. External rewards or extrinsic motivation is motivation based on external factors. While most open source developers don't get paid directly they may receive indirect rewards such as expanding their knowledge or fulfilling their need for particular software.

The results of the different studies [13] & [14] regarding the most important reasons to contribute vary. However, based upon these studies can be concluded that the joy of programming itself forms the most important motivation to participate in an open source project. External factors follow and stand on a slightly lower (if not equal) footplate.

In the context of the Trinidad Platform the external factors are the most important to focus on. The extent to which a developer likes to program is personal and project independent and therefore not suggestible, furthermore most of the potential contributors already possess this intrinsic motivator. External factors however are profitable to examine in order to attract contributors. If we look at external factors according to [13] we can distinguish a number of different categories:

- *Revenues from related products and services*
  As an open source program grows and its usage starts to spread a lot of companies start to offer commercial consulting, training, support etc. This is an important external factor for developers to participate in the Trinidad Platform. As founder of the Trinidad Platform Capgemini will initially provide training and service and collect the revenues. Companies that foresee a grow or see enough potential in the Trinidad Platform can put up developers to participate in order to later on deliver service themselves.

- *Human Capital*
  Developers participating in the development of the framework in order to expand their own skill base. Personal skills, capabilities and knowledge are the forms of capital that are gained here. Companies using the Trinidad Platform will gather knowledge on different fronts because of the many development aspects that are covered by the platform.

- *Self-marketing*
  Developers often use open source projects to demonstrate their programming skills. Sometimes this can work against the open source principle, by commercial vendors luring the most productive minds away from these projects into commercial development. The Trinidad Platform supports the easy integration of components, giving developers the ability to contribute marked pieces of functionality.

- *Peer recognition*
  Open source developers receive rapid and constructive feedback about the quality of their work. A company letting its developers work on the Trinidad Platform can increase the skills of the developers.

- *Personal Needs*
  Many open source projects are initiated or joined because developers have a personal need for software. This is the most important motivation for people to use and participate in developing the Trinidad Platform. In a time where development needs to be faster a standardized reference architecture supported by an extensible framework are key elements in the development of software. Companies are going to want to speed up their own software development by using the

Trinidad Platform. Bugs and additions to the framework will be reported during use, therefore increasing the quality.

If these benefits or future returns are supported in an open source project it makes them interesting for contributors. However these benefits are somewhat objective and every developer may see other benefits in the same project. In order to motivate contributors (being companies adopting the platform for their own development process or developers) the Trinidad open source model needs to contain elements that support or promote these benefits.

3.7 Safeguarding the quality of code

It is often proclaimed by open source proponents that open source development stands for higher quality of code because of the presence of many developers. To what extent is this true, what are the aspects of open source responsible for high quality and how can the quality be safeguarded in an open source project? Stamelos et al [15] have done an extended study on the quality of code within open source projects, using a set of ten metrics among which the McCabe index [16].

As a result of the study there are things that speak for and things that speak against the quality of open source software. An important requirement on open source code is that it needs to be modular, self-documenting code in order to allow development by many people at the same time. Looking at the quality of code, according to [15] the average percentage of acceptable components (functions) across open source programs is high, and half of the components are in good shape. On the contrary the other half of the components are below industry standards in contrast to what open source proponents have claimed up to now.

Commercial vendors often claimed that the quality of code in open source software is weaker, because of the lack of professional testers. However a study of an inspection firm on the code quality of the Apache Web Server concluded that it is comparable in quality to its commercial brethren [17].

Comparing the different facts that are available there can be concluded that the open source code is not of lesser nor higher quality than conventional code.

The question that rises here is why open source software isn't automatically of better quality, after all there are a great number of developers that can fix bugs, perform code-reviews, etc. According to McConnell [18] the strength of open source lies in its massive code-level peer review. However, McConnell states that open source lacks a structured development process which harms the efficiency of working and therefore the quality of code. The best attempt to describe the open source development process is done by Eric Raymond in "The Cathedral and the Bazaar" [19], but according to McConnell even this description can't be called complete.

However, to safeguard the quality of code in an open source project, and also in the Trinidad Platform, there are a few key points to keep in mind [15].

- Develop clear coding guidelines and request from the programmers to keep to this guideline.

- The project co-coordinator can asses the code returned by programmers according to a guideline. This implies that the co-coordinator has the right to reject non-conformant contributions, even if they correct a bug or provide new functionality.

- Code re-engineering decisions can be taken by the project co-coordinator whenever the project seems to experience problems.

3.8 Implementing an open source Strategy

This chapter covers the steps that are involved when implementing an open source strategy. Besides selecting an appropriate open source license and business model there are other steps that need to be taken. The guidelines below are helpful when implementing an open source strategy. A variety of tools are available that simplify many of these steps.

Clean code
Code that will be released under an open source license must be clear of any technology licensed by third parties. Best is to remove the code entirely or seek permission to include the code. To ensure the code is ready to be sent out into the public make sure the code doesn't contain any inappropriate comments, etc.

Core development team
Especially in the early stages of the introduction of an open source project it is necessary that the organization has a team that leads the development of the product.

Setting up a community
Support for external developers: newsgroups, source code repositories, bug-tracking systems. The developers of the core development team should also be a member of the community and develop in the same manner as external developers.

Version control
Different releases of the product will be identified by their associating version numbers. The purpose of assigning version numbers to different releases is to identify the stability and to differ between previous versions.

Change management
Especially in an open source project changes can be multiple and complex. In custom software development a change is always carried out as the result of a change request [20]. In open source software development however, there are no change proposals. Everyone can propose a change and even implement it right away.

Release management
Conducting a release is an important activity within an open source project. Therefore it is necessary to have proper guidelines for release management. Erenkrantz [21] makes the following division in the release management process.

- Pre-release testing; Test a set of standard criteria that every release has to meet. These criteria can vary from code passing unit test to the presence of new features. These criteria can also contain rules about fore- and backwards compatibility.
- Release approval; A release manager, perhaps the head of the core development team, needs to approve a release before it is distributed.
- Distribution; The community needs to be made aware of the new release and needs to have access to the release. The release needs to be in a format that every user can deal with.

4 Successful open source projects

There are a lot of open source projects, some more successful then others. This chapter covers the analysis of two successful open source projects. The open source projects will be analyzed according to the open source characteristics studied in the previous chapter. The way the different processes are organized within these successful projects are analyzed in order to serve as input for the Trinidad open source strategy.

The open source projects that will be analyzed are Hibernate and the Spring Framework, because both are frameworks that speed up development, therefore showing some kind of resemblance to the Trinidad Platform.

4.1 Hibernate

Hibernate is an object / relational persistence and query service [22]. Hibernate's goal is to relieve the developer form 95 percent of common data persistence related programming tasks, compared to SQL. Hibernate represents data in the database as simple Java objects, which provides easy access to data. Hibernate is a free open source Java package, which also exists as a version for the .Net framework named NHibernate. All the information in the chapters below originates from [22].

4.1.1 Open source license

Hibernate is licensed under the terms of the LGPL. This protects the Hibernate community as well as any contributors to Hibernate. Anyone who wanted to release Hibernate under a different license would have to obtain permission from all the contributors who have contributed code to Hibernate, which would be impractical. Therefore Hibernate will remain free software. If Hibernate is changed in any way, those changes should be made available to users, under the same license terms as Hibernate (i.e. the LGPL). [22]

4.1.2 Business model

Hibernate was originally founded by Gavin King in order to solve his own development problems. JBoss Inc. later sponsored the development of Hibernate by employing several members of the Hibernate developer team.

JBoss Inc. clearly uses the Support Sellers business model to make profit out of Hibernate. JBoss employees a team of professional full-time developers dedicated to the project. To cover these costs JBoss Inc. provides Hibernate-specific development and production support contracts with service levels up to 24x7 with a 2 hour response time [22].

4.1.3 Safeguard the quality

In order to safeguard the quality of the project hibernate has taken the key points mentioned in chapter 3 in mind. The Hibernate development team assesses the code returned by programmers and decides whether or not the code is of expected quality. In order to help all the developers reach the expected quality clear coding guidelines are set up.

4.1.4 Community

Hibernate is supported by a wide community which consists of the following elements:

User forum
Forum for Hibernate and NHibernate users, those who use it to aid in their own software development. Users can post questions and answer questions from other users. Posting questions in some of the categories costs 'credits' while answering them in whatever category provides the user with credits. The forum is also used for announcements like new releases, etc. Remarkable about the user forum is that there is a category for a few major languages. In these categories users can ask questions in their own native language.

Hibernate has an extended etiquette on how to ask for help while posting in the forum in order to prevent multiple identical questions and to keep the information clear.

Developer Mailing list
Discussions about the internals of Hibernate and NHibernate can be held in the developer mailing list. Developers can subscribe to the list and enter in discussions on development of Hibernate itself.

Bug tracking and feature requests
Hibernate uses an issue tracking system called JIRA. Users are allowed to browse the database without logging in, but in order to submit, track, and vote on issues, a user must register with JIRA. Even when it comes to submitting bugs Hibernate has etiquette in order to keep information clear. Potential bugs need to be discussed in the user forum before being inserted into JIRA, because according t0 Hibernate 99% of bugs found by users are actually misuse. Besides bugs feature request and patches can be submitted into JIRA as well. Patches need to be submitted in 'patch' format.

Team weblog
There is a central weblog for Hibernate where members of the team recently post new items.

4.1.5 Version control

Since of March 2006 Hibernate's source is maintained in a subversion repository hosted by the JBoss Labs project. Besides the Hibernate core the subversion contains the source of a lot of Hibernate add-ons like Hibernate Tools, Hibernate Entitymanager and others which are all also available for download at [22].

Backwards compatibility
Hibernate strives for backwards compatibility but doesn't fully guarantee this to its users. Hibernate deals with compatibility or migration problems by releasing a migration guide with every new major Hibernate release. This guide specifies in detail which functions, methods or classes are left out or modified and therefore might lead to compatibility problems. Based on this guide users can determine whether it is smart to migrate to the new version or not.

4.1.6 Change management

Hibernate offers anonymous and developer access to the source code in subversion. Anonymous access doesn't contain rights to commit. Contributors can check out the latest Hibernate version from subversion, but can't commit modified source code. Patch files need to be made of the changes so that they can be submitted through the JIRA issue tracking system. The benefit of this approach is that al the changes and bug fixes arrive through one central place.

There is however a group of developers that do have rights to commit to the Hibernate subversion source code. These developers are members of the Hibernate team. The Hibernate team gathers all the patches submitted through JIRA. These patches are then assessed and applied or rejected.

4.1.7 Release management

There is a release checklist for Hibernate developers that have the rights to issue a new release. This list describes the steps that need to be taken in order to issue a release [22].

Pre-release testing
Unfortunately the list doesn't specify anything about running regression tests. All the submitted patches need to be accompanied by TestFixtures so it can be assumed that these test are run before the release is built.

Release approval
The approval of releases is done by the Hibernate development team. However there is no information available about how this process is organized and who has the final responsibility for a release.

Distribution
Hibernate is distributed in the form of an ant distribution. Before distribution the changelog, readme and other files need to be modified according to the release checklist. The new release is announced on the forum, the developer mailing list and the front page [22] accompanied by a list of the most important changes. The releases can be downloaded from the Hibernate web site.

4.1.8 Remarks

Hibernate is an open source project that can be called successful. Hibernate is more than five years old, the Hibernate core is over 76 000 lines of Java code, together with 36 000 lines of Java unit test code. About 25 000 developers are registered on the Hibernate forums (including international forums for German, French, Chinese, and Russian speakers).

The main development and course of Hibernate is decided upon by the Hibernate development team employed by JBoss. The source code is freely available to everyone. Developers can edit the source locally in order to solve bugs and contribute their changes, but they will be assessed by the Hibernate development team.

One thing that Hibernate lacks is documentation about the internals and the most important objects that Hibernate contains. Besides the API javadoc there is no concrete description of the basic toolkit a user needs in order to implement persistence in an application.

4.2 Spring Framework

The Spring Framework [23] is a layered Java/J2EE application framework based on code published in "Expert One-on-One J2EE Design and Development" by Rod Johnson [24]. The Spring Framework provides solutions to many technical challenges faced by Java developers and organizations wanting to create applications based on the Java platform [25]. The Spring Framework was first released as an open source project in February 2003.

4.2.1 Open source license

The Spring Framework is licensed under the Apache License, version 2.0. This basically means that modifications and add-ons to the code can be kept secret and or sold. Using this license carries the risk that all the modifications don't come back to the original version.

4.2.2 Business model

The initial Spring Framework was developed by Rod Johnson while writing his previously mentioned book. Because of the potential the framework possessed an open source development team was formed in order to further develop the framework. There was no initial Business model used, as the framework was developed for own use. Interface 21 however, the company Rod Johnson is currently working for is using the Support Sellers business model. The company offers commercial support on and training in the Spring Framework.

4.2.3 Safeguard the quality

The Spring Framework has a way similar to Hibernate in order to safeguard the quality of their code. They have a core development team that asses the code that is contributed by developers and also have a list of coding guidelines to assist the developers.

4.2.4 Community

The Spring Framework is also supported by a community, consisting of the following elements:

Spring Framework Support Forums
The Spring Framework contains a lot of categories of development topics. There is even a Chinese support forum for the Chinese speaking Spring Users. Spring doesn't have such strict rules regarding posting topics. The only etiquette mentioned is to search for the answer first in order to save time.

Spring JIRA Issue Tracker
Just like Hibernate the Spring Framework also uses the JIRA issue tracker. Bugs and feature requests can be submitted, even as patches to bugs. There are a few guidelines regarding to posting bugs, (like searching if the bug isn't reported earlier) but they are not as strict as the ones Hibernate handles.

Developer Mailing list
Developers of the framework itself can subscribe to the Spring Framework developer mailing list. This mailing list contains discussions about the internals of the Spring Framework.

4.2.5 Version control

The source of the Spring Framework is available in a CVS repository at SourceForge. The server can be reached through anonymous access. Only for project-developers is the code available through developer access. Developers can edit the code directly from the CVS while through anonymous access the code can only be checked out.

Backwards compatibility
The Spring Framework developers feel very strongly about backwards compatibility. All the Spring Framework releases claim to be 100% compatible with their previous versions. This prevents users from migration problems, but considering previous releases can get more troublesome as the project grows and knows more historic releases.

4.2.6 Change management

The spring Framework knows a change management process similar to Hibernate. Users can apply patches to bugs that will be picked up by a member of the Spring Framework development team. The patches will be assessed and rejected or applied.

4.2.7 Release management

Unfortunately there is not much information available about the release management process of the Spring Framework. However, the fact that the nightly snapshots of a release candidate are distributed for testing and development purposes before the release is distributed leads to believe that there is a more sophisticated pre-release test process underneath, but this isn't specified anywhere. Neither is information about the approval of new releases.

New releases are announced in the support forums, the Spring Framework home page and on the developer mailing list.

4.2.8 Remarks

The Spring Framework was first released as an open source project in February 2003. Because of his young age the Spring Framework is not as mature as Hibernate. The amount of developers and users involved in the Spring Framework are less than those of Hibernate. This can be seen back in the “loose” guidelines the Spring Framework handles in comparison to Hibernate.

5 Success factors

Looking only at how the open source projects from the previous chapters manage and or handle the open source characteristics isn't sufficient for drawing conclusions and issuing an advice for the Trinidad open source project. In order to get a complete picture of the reason for their success research needs to be done about open source success factors in general, and the specific success factors for these projects.

For a project to contain all the open source characteristics that were discovered during research and to manage them in a proper way can be considered a success factor itself. If an open source project would take all these characteristics into consideration, like choosing the right business model, the right license, having a clear version and configuration management, etc. the project would have a strong basis. However just managing these characteristics does not guarantee a successful open source project. On their site [22] Hibernate points out a few factors that helped make the project popular and successful. Unfortunately the Spring Framework didn't release such a documented list of success factors. However many of the factors below are also present in the Spring Framework.

- *Rapid release schedule;* Regular file releases (even a few days from one version to the next) are the best way to keep the project bug-free and users confident that the project is active.
- *Regression tests;* A comprehensive test suite is central to the maintainability and stability of the Hibernate project, as it went through massive changes in functionality and design. The mentality should be that if there's no test for a feature, then we have absolutely no idea whether that feature works or not.
- *Do one thing well;* Be the best at something, let other projects worry about all the other things which you can't be the best at.
- *Avoid over-design;* Aiming for too much abstraction and flexibility at an early stage is a great way to waste time that could be better spent solving actual problems that your actual users are facing.
- *A central vision;* It's got to be a pretty damn big project before you need a committee to make decisions. Most projects are better off with one or two open minded people guiding the project and maintaining a single coherent vision.
- *Documentation;* There's no such thing as an undocumented feature. If your users don't know about a feature, it's a non-feature. Get rid of it; it's just complicating the source code.
- *Avoid standardism;* Good standards can provide interoperability and portability. Bad standards can stifle innovation. Standards are only useful if they are adopted industry-wide.
- *Up and running in ten minutes or less! ;* Prospective users don't have time to spend half an hour installing, configuring, troubleshooting a piece of software the first time they download it. Hibernate's aim is that new users can have the demo running in five minutes (assuming sufficient knowledge of JDBC) and their own simple Hello World style application going inside of an hour.
- *Developer responsiveness;* When users have problems, which they inevitably will, the development team must be responsive and helpful. Users let you know about holes in your documentation. Users find the subtle bugs that your test suite misses. And besides, the project is a waste of time without them.
- *Easily update-able wiki pages.*
- *Professional Open Source;* Not many business models have emerged to allow open source developers to make a living from their software, one of them is JBoss Inc's Professional open source. In addition to supporting developers, it allows business users to get what they need in critical projects: development support, production support, and professional training. A successful open source project needs full-time developers and commercial backing.

5.1 Literature

Besides the last success factor all of them are in terms of development practice. Having a strong community is just as important as good development practice. This leads to believe that success factors can be divided into development success factors and success factors that improve the collaboration and communication within the community. In [26] the authors already made such a distinction between success factors for the community and success factors for the software development, which are shown below.

Factors for a successful community:

- To establish a strong network there must be active marketing and promotion of the open source project.
- Members of open source projects are usually spread all over the world; they need a central place of appointment with support fora and communication amongst all community members.
- Ideas and viewpoints of others in any respects must be accepted, appreciated and incorporated, if appropriate.
- Stability, openness, transparency and fast response times for these communication and information exchange activities.
- Conferences, developer meetings and workshops about a special subject.
- A sense of community together with clear dispute resolution mechanisms – in a democratic sense of decision making.
- Initiation of self-organization processes like support of new participants by experienced community members.

As a conclusion to these factors the study in [26] states that a flat organization cannot fit the needs for coordination, guidance and involvement in many respects. This means that there must be project management!

Factors for successful open source development:

- Short intervals for new releases and application and testing by as many users as possible (predictability).
- A clear declaration and identification of beta-and stable releases.
- A comprehensive documentation of the code and a roadmap for development.
- General preparations (by a core team) for and following discussion of requirements and targets of further development in the community.
- Definition of preferences and priorities of certain projects.
- Avoidance of monolithic code.
- Permanent quality management.
- A comfortable opportunity for distributed software-development with a concurrent versions system.
- Permanent bug-fixing

As a closing conclusion to these factors the following can be stated: Besides project management there must be a competent technical core of developers who guarantee ongoing development, quality of the product and support in a growing community!

Besides community and development factors, economic aspects cannot be neglected if an open source project wants to be successful. Open source projects, just like closed source projects, need some kind of business model.

The subject of business models has already been fully studied as part of the open source characteristics. Looking at the rest of the success factors some of them are already mentioned under the open source characteristics. However like mentioned earlier good management of the open source basics is a success

factor itself. Therefore it's not strange that many projects and studies point these characteristics out as success factors.

Most of Hibernate's success factors from the previous chapter can be mapped directly to the ones in this chapter. The success factors that can't be associated with any factors within this list, and are general enough are added. The combined list of success factors is used for the rest of the research.

5.2 Hypothesis

Based on the above success factors and the research done the following hypothesis can be drawn.

"If an open source project manages all the characteristics in a way that is appropriate for that specific project and that project takes the success factors into consideration the process will result in a successful open source project."

In order to check the correctness of this hypothesis research will be done about failed open source projects, and their reason of failure. The thought behind this research is to check if application of these open source factors could have prevented failure.

5.3 Succes factors of Hibernate and Spring

Before studying the failed open source projects it is interesting to see to which extent the two successful open source projects from the previous chapters support these succes factors.

OS projects / **Succes factors**	Hibernate	Spring Framework
Community		
C1.Marketing / Promotion of the open source project	+\-	+\-
C2.Central community with support fora and communication	++	++
C3.Respecting and accepting ideas and viewpoints of others, if appropriate	+	+
C4.Stability, openness, transparency and fast response times for these communication and information exchange activities	?	?
C5. Conferences, developer meetings and workshops about a special subject.	+	++
C6.A sense of community together with clear dispute resolution mechanisms – in a democratic sense of decision making.	?	?
C7. Initiation of self-organization processes like support of new participants by experienced community members.	?	?
C8. A central vision, most projects are better off with one or two open minded people guiding the project and maintaining a single coherent vision.	++	?
Development		
D1. Short intervals for new releases and application and testing by as many users as possible	++	?
D2.A clear declaration and identification of beta-and	++	++

stable releases.		
D3.A comprehensive documentation of the code and a roadmap for development	+	+
D4. General preparations (by a core team) for and following discussion of requirements and targets of further development in the community.	?	?
D5. Definition of preferences and priorities of certain projects.	+\-	+\-
D6. Avoidance of monolithic code.	+	+
D7. Permanent quality management.	++	++
D8.A comfortable opportunity for distributed software-development with a concurrent versions system.	++	++
D9. Permanent bug-fixing	++	++
D10. Avoid over-design, Aiming for too much abstraction and flexibility at an early stage is a waste of time.	+	?
Economical		
E1. Professional Open Source, generating revenue by choosing the right business model.	++	++
E2. Full-time developers backed by a commercial company.	++	++

There are a few unknown factors from which it is not clear whether they are present in Hibernate and the Spring Framework or not. However, leaving these unknown factors out of consideration the matrix shows that these two projects score well on every success factor. This result speaks in favor for the defined hypothesis. In order to strengthen this conclusion the following chapters discuss two failed open source projects.

5.4 (Open)Darwin

Since its first release in March 1999, Darwin has been the open-source OS technology underlying Apple's Mac OS X operating system. All development was being managed and hosted by Apple. Darwin was a free UNIX based OS for Mac users and Apple's Mac OS X releases were based directly on the live Darwin CVS repository. In order to improve cooperative development OpenDarwin [27] was cooperatively founded in April 2002 by Internet Systems Consortium, Inc (ISC) and Apple. Many of the members of OpenDarwin were either Apple employees or Darwin Committers.

5.4.1 Apple's motivation

When Apple started shipping Mac OS X Server, it included some GPL'd projects in addition to many other open source projects. Apple thought that since they were required to release some of the source anyway, why not release modifications to all of their open source software? From there it was one short marketing leap away from releasing the low level components of Mac OS X as open source, and one large, consistent project to release the entire base of Mac OS X as a standalone OS. This was Apple's way of getting a piece of the increasing open source pie. However, just releasing some source code wasn't enough. Apple needed something concrete they could show. As an answer to this Apple came up with a build system that was compatible with Apple's internal build system so they could offer the open source community, a new, free operating system called Darwin.

5.4.2 Darwin open source project lifecycle.

Rob Braun, an employee of Apple during the Darwin open source project elaborately discusses the failure of Darwin on his webpage [28]. All the information in this chapter origins from this website. At the start

the Darwin open source project showed much potential. Contributors were excited by the possibility of a free OS for their MAC, others just to be part of a brand new project. Being supported by a large company such as Apple, promised great things for Darwin.

However the potential was slow to be realized. With the release of Darwin 1.2 also came Mac OS X Public Beta. There it came to light that Apple cared more about the product Mac OS X than Darwin and the community. Some of the Darwin source started having dependencies to non-open source components. Apple started to work as a whole on Mac OS X while Darwin got engineering leftovers. This could be seen back in unusual decisions for the open source project, like removing old hardware support. Clearly this was done to Mac OS X as a business decision, so Darwin automatically inherited this.

Apple was developing a reputation for not taking contributions, particularly not new features or additions. Small bug fixes which were easily reviewed might be accepted back, but medium or large sized features were not getting traction which upset the developers. During this time Apple ended up hiring over half of their most active contributors, which drained the contributor pool significantly. The Darwin lists were almost entirely populated by people running Mac OS X and had questions about UNIX problems.

Apple recognized the problems and hired people to create a better Darwin community. This resulted in the OpenDarwin project, jointly created by Apple and ISC.

With the creation of OpenDarwin Apple was able to push back open source and satisfy with hardware donations instead of cultural shift and true interaction. However the original goal of creating a free standalone OS and a community around it had failed. There was no community, and the free standalone OS had become composed of binary packages and source no one could build. On the 25th of July 2006 the OpenDarwin project was shut down.

5.4.3 Reasons for failure

After analyzing the project lifecycle and the failure of Darwin the following reasons for failure were drawn up:

- Marketing trick; Apple's marketing department thought they could increase their market share and revenue by going open source. Apple was right with this. The popularity of Mac OS X has been growing since. However Apple realized this goal at the cost of Darwin. It seems like Apple itself, with the exception of a few engineers, had no intention of becoming involved with any community outside its own walls. This point of failure indicates that a good and clear strategy and support from the supporting company is mandatory for an open source project. (Success factors **E1**).

- Community; In order to keep contributors and users pleased they must feel respected and feel that their work is being appreciated. A community plays a great role in pleasing the developers. They need to be able to interact with each other and the core development team. Darwin lacked many of the success factors regarding a strong community amongst which are: Stability, openness, transparency and fast response times, Accepting and appreciating Ideas and viewpoints of others. (Success factors **C3** and **C4**).

- Core development team; In order to achieve fast responses and to appreciate the contributions of developers a core development team needs to exist. This team's only focus needs to be development of the open source project. Darwin did have such a team, however the focus of Apple was more on Mac OS X than it was on Darwin. (Success factor **E2**).

- Motivation of developers; In order to keep a project alive the developers need to be motivated. However Darwin developers were either hired by apple or left the project with dissatisfaction.

The failure of Darwin lies for a great deal in Apple's marketing concerns (Mac OS X) and the inability of Apple to set up a community around their project.

If we compare the reason of failure and the total process of Darwin to the list of open source success factors there are a few things that can be noticed.

The open source project started out with a bad basis, overall management of the open source characteristics wasn't done properly. There were a lot of hurdles, like registration, to overcome in order to download the software. After download the project was hard to build. Despite these hurdles some people who saw the potential in this project accepted these "start-up" problems and were excited about the project

Most of Darwin's failure factors are focused on the community. The community lacked a lot of essential things, already mentioned with the reasons of failure. Only a few of the success factors regarding a community that were stated in the previous chapter can be found back in the Darwin open source project.

As the list of success factors concludes economic aspects cannot be neglected if an open source project wants to be successful. However the marketing strategy and chosen business model of Apple didn't fit an open source project. Apple chose to create revenue from their commercial product Mac OS X by releasing an open source derivative. Because of the model the revenue generating product received the most attention which led to failure of the open source project.

Concluding it is important to remark that all the points responsible for the failure of Darwin were in the previous chapters identified as open source project characteristics that need to form the basis of every open source project and open source success factors. This indicates that if Apple had set up its open source project according to the characteristics and with the success factors proposed here, Darwin probably wouldn't have failed. These results speak for the correctness of the open source success factors.

5.5 NDoc

Compared to Darwin Ndoc[29] is a smaller open source project. Ndoc generates class library documentation from .NET assemblies and the XML documentation files generated by the C# compiler. Ndoc is created by Kevin Downs, who developed the project in his spare time. Ndoc isn't backed up by a commercial company. Therefore the motives for the creation of the Ndoc project were simple, the tool was created for personal use and the developer wanted to share it with the world.

5.5.1 Ndoc project lifecycle

Before his departure Kevin Downs sent an email to the NDoc community, explaining the reasons for the termination of NDoc. This email can be read on a lot of blogs on the internet amongst which the one of B. House [30]. The information about the NDoc project lifecycle in this chapter origins from this email and the NDoc website [29].

According to many developers Ndoc is more or less being considered to be the standard for documenting .NET class libraries. Still the creator of the project didn't get any development help from the community. As Ndoc started to grow in popularity, so did the user base. However the project failed to attract developers who wanted to participate in development of the product itself.

When Ndoc started to become popular and was being used and depended upon by people, the community started to act like Ndoc was commercial and demanded to receive updates and fixes. The developer of the project, Kevin Downs, started to feel more and more pressure from the community and less appreciated. Combined with the lack of donations this made him announce the end of the project on the 26th of July 2006.

The closure of the project is sad news in the development community. However the retiring developer is opening up the position of project admin these chances that some one else will take over are considered to

be small. Remarkable is that many of the developers using Ndoc state that the Ndoc project has a lot of potential to go commercial, and if it did, they would be eager to pay for it.

5.5.2 Reasons for failure

With his departure Kevin Downs mentioned a few reasons why he left the NDoc open source project [30]. All these reasons can be traced back to the following reasons of failure which are stated below:

- Lack of support; The main reason the NDoc project failed is the lack of support and appreciation from the community. Everyone was using the software without contributing anything back. The project failed to attract developers willing to participate in development. (Success factor **C1**).

- Core development team; the project received bug reports and feature request but unfortunately these were too much for the core development team. In this case the core development team existed only of one developer working on the project in his spare time. The user base that was starting to depend on NDoc got impatient. (Success factors **D1**, **D9** and **E2**).

Comparing the problem in this project to the list of success factors the crucial factor NDoc lacked is having a core development team stable enough to withstand the pressure from the community. Every core development team would, at one moment in time yield under the pressure of the growing user base. However a 10 person core development team has a longer time span to this period than the one-man core development team of NDoc. Within this extra period of time the possibility that developers attracted to contribute instead of only use the project would come by is much larger.

NDoc yielded under the extra risk that an open source project not backed up by a company is facing; once the project becomes successful the success totally depends on the willingness of the users to either contribute to development or to donate to the project.

The second failed open source project that was studied didn't test much more of the success factors than the previous. The importance of having a stable core development team has become clear however, with that supporting the correctness of this open source characteristic.

Concluding to the research on the failed open source projects, they are compared below in the success factor matrix.

OS projects / **Succes factors**	(Open)Darwin	NDoc
Community		
C1. Marketing / Promotion of the open source project	--	--
C2. Central community with support fora and communication	--	-
C3. Respecting and accepting ideas and viewpoints of others, if appropriate	--	+
C4. Stability, openness, transparency and fast response times for these communication and information exchange activities	--	--
C5. Conferences, developer meetings and workshops about a special subject.	?	--
C6.A sense of community together with clear dispute resolution mechanisms – in a democratic sense of decision making.	-	?
C7. Initiation of self-organization processes like support of new participants by experienced community members.	?	?

C8. A central vision, most projects are better off with one or two open minded people guiding the project and maintaining a single coherent vision.	-	+
Development		
D1. Short intervals for new releases and application and testing by as many users as possible	-	--
D2.A clear declaration and identification of beta-and stable releases.	+	+/-
D3.A comprehensive documentation of the code and a roadmap for development	-	+/-
D4. General preparations (by a core team) for and following discussion of requirements and targets of further development in the community.	?	?
D5. Definition of preferences and priorities of certain projects.	?	?
D6. Avoidance of monolithic code.	?	?
D7. Permanent quality management.	+	+
D8.A comfortable opportunity for distributed software-development with a concurrent versions system.	++	++
D9. Permanent bug-fixing	-	-
D10. Avoid over-design*;* Aiming for too much abstraction and flexibility at an early stage is a waste of time.	?	?
Economical		
E1.Professional Open Source, generating revenue by choosing the right business model.	--	--
E2.Full-time developers backed by a commercial company.	+/-	--

The matrix shows that, besides the unknown factors, these two projects lacked a lot of success factors which probably caused them to fail. However this matrix once again speaks in favor for the defined open source factors and the hypothesis.

5.6 Evaluation

The list of success factors couldn't be proven incorrect by analysis of both of the failed open source projects, their correctness was rather indicated both times. Both projects failed by lacking some basic open source characteristic and/or an open source success factor.

The research has led to the understanding that the standard open source characteristics are success factors themselves. It seems that proper management of the open source characteristics when starting an open source project provides a solid basis for a successful project. Many projects fail because of bad management of these characteristics.

The result of the analysis of two failed open source projects indicates that a combination of the open source characteristics and the list of open source success factors lead to a successful project. Based on these findings the formulated hypothesis can said to be correct.

However in order to be able to give a more grounded judgement on the correctness of the hypothesis the following activities could have been done different:

- Full research on failed projects; In the current research only the failure factors of the failed open source projects have been analyzed. To get a better view of the failure of the open source project the open source characteristics need to be researched. The way the project managed these characteristics and to which extent are important factors that affect the failure or success of the project.

- Assessment of all the success factors; The current research tests the open source factors according to the failure of two open source projects. The way in which the success factors could have prevented the failure of the project is analyzed. However, this analysis focuses on the specific success factors responsible for prevention of the project failure. The rest of the factors in the list are judged upon in the matrix, but not fully researched and tested. Additional failed open source projects are needed in order to fully test the complete list of success factors.

The total open source research has resulted in the Trinidad open source advice which is described in the next chapter.

6 Trinidad open source advice

This chapter will contain a grounded advice based upon the research described in chapter 4 and experiences from other open source projects as described in chapter 5. This advice will form the strategy for an open source Trinidad Platform.

6.1 License and Business Model

Users or developers of the Trinidad platform might fix bugs or develop additional features. It is important for Capgemini that these modifications are contributed back into the framework so everyone can benefit from a better working framework. Capgemini can enforce this by using the right open source license.

As the study resulted there are a lot of open source licenses available. The Spring Framework uses the Apache license, which allows the users to keep modifications private. Developers of the Spring Framework can make modifications and for example easily sell them. This is not what Capgemini wants for the Trinidad Platform. In order to suit the needs of Capgemini best, the most appropriate license to be chosen is the LGPL license. This license is also used by Hibernate and enforces users to distribute modifications under the same license.

Another characteristic of the LGPL license is that software licensed under it can be mixed with non-free (commercial) software. Because of the open and modular structure of the Trinidad reference architecture it is not unthinkable that the need for connecting with commercial software might occur. The LGPL license is flexible enough to allow this. The full description of the LGPL license can be found at http://opensource.org/licenses/.

Business Model
In order to create revenue from an open source model it is wise to decide on a business strategy. Capgemini wants to offer customers the Trinidad Platform for free but still generate revenue. The approach that Capgemini wants to take in this is to generate revenue by offering training and support.

The companies behind Hibernate and the Spring Framework have business goals similar to Capgemini's. These business goals are accomplished by using a business model called "Support Sellers". These companies have employed a core development team, devoted to working on the project. They provide professional support and training to users and developers of the product. This business model has the advantage that companies using the open source software rather turn to the original developers for support. The more successful the open source project becomes, the more income the business model generates.

Capgemini needs to carry out the Support Sellers business model in order to meet their business goals. Because of the nature of the Trinidad Platform it is likely that the platform will be mostly used by companies rather than individual developers. Especially for companies it is important that the open source project they use is backed up by a company. This gives them a sense of security. When in need of training and support Capgemini will be the obvious company to provide this.

6.1.1 Core development team

In order to support the Trinidad Platform Capgemini will need to form a core development team. Among other things this team will be responsible for releasing new versions of the Framework and coordinating the development of the Framework. The other functions this development team will fulfill will become clear in the following chapters.

6.2 Developers

As we have seen developers are the most important factor in any open source project. Without developers willing to participate and contribute time and effort in an open source project it will surely fail. The study in chapter 3.6 results in a few external factors for developers to contribute to an open source project. Some of these external factors are of a more objective personal nature and can not be influenced by a specific project (like peer recognition). Therefore the most a company can do to make developers comfortable is to provide or stimulate these external factors. In the context of Capgemini and the Trinidad Platform the following issues need to be emphasized in order to make the project interesting for developers and / or customers.

Ready for the Future
Development of software needs to be done faster and faster, because of the growing demands of the customers. In order to reach faster development the development process needs to be standardized. Development within the Trinidad Platform is highly standardized and contains elements that speed up development. Working on a new way of development can boost the joy of programming for developers, and make it interesting for companies to learn the framework and later on even deliver support.

Expanding the developer's skill base
Development within the Trinidad Platform contains a lot of modern development techniques like Use Case modeling, Use Case estimation, Model Driven Architecture and an agile development process. Developers can gather a lot of knowledge on different techniques which make it interesting to participate.

Developer's recognition
Developers like to show their skills, and even get recognition. A reward system within the community could be a good way to support this. Hibernate has a system where users get 'credits' for answering questions from other users. A similar system, with rewards for bug-fixes and patches can attract developers.

Personal needs
Every software development company needs a structured development process. Especially for companies it is interesting to adopt and use the Trinidad Platform.

6.3 Safeguard the quality

As chapter 3.7 already states the strength of code quality within open source projects lies in its massive code-level peer review. However open source projects lack a structured development process which hurts the quality of code. The benefit of the Trinidad Platform is that it contains a structured development process itself. Development of the framework itself needs to be done according to the same agile development process. Sticking to this development process and keeping in mind the key points below can safeguard the quality of code within the Trinidad platform.

- Develop clear coding guidelines and request from the programmers to keep to this guideline.

- The core development team could asses the code returned by programmers according to a guideline. This implies that the co-coordinator has the right to reject non-conformant contributions, even if they correct a bug or provide new functionality.

- Code re-engineering decisions can be taken by the core development team whenever the project seems to experience problems.

6.4 Community

A successful open source project needs a community where developers can get in touch with each other and provide support amongst them. Users need to be able to ask questions and be kept up to date about news and info regarding the project.

Trinidad User Support Forum
Looking at the Hibernate and the Spring Framework we can learn that it is advisable to distinguish between user support and framework developers support. The user support forum will allow users to posts and answer questions about Trinidad based software development. A reward system similar to the one Hibernate uses is a valuable extra function to the user forum. It is also preferred that the forum supports multiple languages in order to support users from all over the world. However the initial forum needs to be English, and can later on be extended with additional forums in other languages.

To keep postings in the forums structured it is wise to create a forum etiquette containing guidelines on how to post a question, the amount of additional information that needs to be posted along with the question, etc.

Developer's Mailing list
Many open source projects (Hibernate and the Spring Framework) use a mailing list for developers support. Mailing lists have the benefit that developers stay updated about everything that happens, because every mail that is submitted is received by every member.

It is wise to set up a mailing list within the Trinidad Platform. The Havana framework developers can discuss technical issues in the Trinidad Mailing list. The mailing list is intended for use only by developers actually working on development of the Havana Framework.

Bug tracking
Using a bug tracker is mandatory for every open source project. There are many open source and commercial bug trackers available. The bug tracker is the place where bugs, patches to bugs and feature requests can be submitted. Also for reporting bugs there need to be clear guidelines. Because the Trinidad platform is a new open source project there aren't as many developers reporting bugs yet. Therefore the guidelines don't need to be as strict as for example the ones Hibernate handles. As the project and the amount of developers grow the guidelines can be adjusted if necessary in order to reduce 'traffic' in the bug tracker.

Portal
The Trinidad open source project needs a project portal where users can find information about the Trinidad platform. Documentation like coding guidelines, manuals and news items need to be available on this site. Also all the other aspects of the community (forum, bug tracker) need to be present on or through this site. As the company supporting the Trinidad development Capgemini needs to host and maintain this site. Possible courses and training regarding Trinidad can also be offered through this portal.

6.5 Version control

The source code of the Havana Framework needs to be made publicly available in order to become a full open source project. However, releasing the source code doesn't mean just offering any body access to modify the code. The source code needs to be placed in a repository (CVS, Subversion, etc) so multiple developers can work together. A good strategy used by Hibernate and the Spring Framework is to make a distinction in rights between developers and users. Developers are initially the core development team from Capgemini containing rights to check out and commit modifications to the source code. In the light of safety and safeguarding the quality of code everyone else has user access. This means that the code can be checked out but modifications can't be committed. Bug-fixes or patches need to be submitted through the bug tracker.

As the project starts to grow the core development team will eventually become too small to keep up with the growth. In this situation the core development team may select loyal users who have committed much to the project through the bugtracker to join the development. These developers will then be part of the core development team and will be given developer rights on the source code.

Backwards compatibility
Open source software doesn't require a company to guarantee backwards compatibility. However it is wise for developers not to vary too much between releases because this will result in dissatisfaction from the users. It is wise to strive to backward compatibility, but guaranteeing 100% compatibility between every release will almost be impossible in the long run. The Spring Framework has all compatible releases up until the present, but hasn't guaranteed this for the future.

It is good for Capgemini to take this as a starting point for the Trinidad Platform. It is wise to strive to maximal backwards compatibility. However if this gets to complicated, mostly when the project starts to grow and expand the strategy of Hibernate can be used. Compatibility and migration problems can be dealt with by means of a migration guide. In this guide all the differences between releases and steps to take when upgrading need to be explained. This doesn't give the users 100% backwards compatibility but compensate for this by guiding the users through the migration process.

6.6 Change management

The proposed change management process for the Trinidad Platform is already partially enforced by the version control system. Only members of the core development team can directly modify the code. All the changes, bug-fixes or feature requests from other developers are submitted through the bug tracking system. The core development team will decide on the correctness of the bug-fix, the quality of the code and reject or approve the change accordingly.

Both Hibernate and the Spring Framework use the same change management process. This is a safe way to secure the integrity and quality of the code.

6.7 Release management

In order to frequently issue new releases of the framework a release management process needs to be configured. Hibernate has a checklist that developers need to follow before issuing a release. The Spring Framework doesn't seem to have a fully structured process. However in both projects the core development team is responsible for the release. This team decides on for example the bugfixes that will be implemented in the next release.

Pre-release testing
Before publishing a new release it should be tested according to a few criteria like code tests, acceptance test, etc. The core development team needs to make sure that every new feature or bugfix is tested. The advice here is to set up guidelines for testing that can be directed back to the developers. Every developer needs to bundle a patch or bugfix with a test.

This doesn't mean that the core development team doesn't need to test anymore. It is the responsibility of the team to gather and structure all the tests, carry out regression tests and perform the final check. Acceptance tests can also be part of the pre-release testing process.

Release approval
A member of the development team needs to coordinate the pre-release activities en give the final approval for issuing a release. This member can function as a controlling mechanism on top of the pre-release testing process. It is not clear how this process is done in Hibernate or the Spring Framework but for the Trinidad Platform it would certainly be a valuable addition.

Distribution
After a release is formed it needs to be distributed. The latest version of the code can of course be obtained from the source code repository, but it also needs to be made available in a more user friendly manner. The first step that needs to be taken is to notify users and developers of the new release. The best place to announce the release is in the community; within the forum, the developer mailing list, the frontpage of the website, etc. Like Hibernate and the Spring Framework the best place to distribute the framework is the Trinidad Platform website.

6.8 Success factors

In order to set up a successful open source Trinidad Platform the open source success factors that have been studied need to be taken into consideration. The open source success factors are described in chapter 5. Some of these open source success factors are closely associated to the open source characteristics that are discussed in this chapter. However the advice to Capgemini is to use the full list of open source factors as a supplement to the open source characteristics.

The advice as presented to Capgemini can be read in the Trinidad open source Strategy document which can be found as appendix B to this document.

Bibliography

[1] Kruchten, P. *The Rational Unified Process: An Introduction, 2nd Edition.* Addison-Wesley, 2000.

[2] Kruchten, P. *Architectural Blueprints – The "4+1" View Model of Software Architecture,* IEEE Software, 1995.

[3] Bass, L., Clements, P., Kazman, R. *Software Architecture in Practice, Second Edition*, Addison-Wesley, 2003.

[4] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns, Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional, 1995.

[5] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Gang of Four Template,* http://hillside.net/patterns/DPBook/DPBook.html, 1995.
Visited at 15-08-2006.

[6] Fowler, M. *Patterns of Enterprise Application Architecture,* Addison-Wesley Professional, 2002.

[7] Mulgan, G., Steinberg, T., Salem, O. *Wide Open, Open source methods and their future potential,* Demos, 2005.

[8] Perens, B. *The Open Source Definition*, http://perens.com/Articles/OSD.html.
Visited at 15-08-2006.

[9] Brooks, F. *The Mythical Man- Month,* Addison-Wesley, 1975.

[10] Hecker, F. *Setting Up Shop: The Business of Open-Source Software,* IEEE Software, 1999.

[11] Wilson, G. *Is The Open-Source Community Setting a Bad Example ?,* IEEE Software, 1999.

[12] Daly, P. *The Economics of Open Source Software,* The University of Waterloo, 2002.

[13] Hars, A., Ou, S. *Working for Free? – Motivations of Participating in Open Source Projects*, IEEE, 2001.

[14] Lakhani, K., Wolf, R. *Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects.*

[15] Stamelos, I., Angelis, L., Oikonomou, A., Bleris, G. *Code quality analysis in open source software development,* Information Systems Journal, 2002.

[16] McCabe, T. *A complexity measure,* IEE Transactions of Software Engineering, 1976.

[17] Boulton, C. *Open Source, Proprietary Code Quality Comparable,* http://www.internetnews.com/dev-news/article.php/2230481, 2003.
Visited at 15-08-2006.

[18] McConnell, S. *Open source methodology: ready for prime time?,* IEEE Software, 1999.

[19] Raymond, E. *The Cathedral and the Bazaar,* http://www.catb.org/esr/writings/cathedral-bazaar/, 2000.
Visited at 15-08-2006.

[20] Asklund, U., Bendix, L. *Configuration Management for Open Source Software*, Aalborg University Dept. of CS, 2001.

[21] Erenkrantz, E. *Release Management Within Open Source Projects*, Proceedings of the 3rd Workshop on Open Source Software Engineering, 2003.

[22] JBoss Inc. *Hibernate*, http://www.hibernate.org.
Visited at 15-08-2006.

[23] Interface21, *Springframework.org*, http://www.springframework.org.
Visited at 15-08-2006.

[24] Johnson, R. *Expert One-on-One J2EE Design and Development*, Wrox Press, 2002.

[25] Wikipedia. *Spring Framework (Java)*,
http://en.wikipedia.org/wiki/Spring_Framework_%28Java%29
Visited at 15-08-2006.

[26] Koch, K., Hartung, M., Hesser, W. *Success Factors of Open Source Projects*, Presentation on the 2nd International ILIAS Conference, 2003.

[27] *OpenDarwin Wiki*, http://wiki.opendarwin.org/index.php/OpenDarwin.
Visited at 15-08-2006.

[28] Braun, R. *Why Darwin Failed*, http://www.opendarwin.org/~bbraun/osfail.html, 2006.
Visited at 15-08-2006.

[29] *NDoc online*, http://ndoc.sourceforge.net/.
Visited at 15-08-2006.

[30] House, B. *Brenton House : NDoc 2.0 – R.I.P.*,
http://weblogs.asp.net/bhouse/archive/2006/07/26/NDoc-2.0-_2D00_-R.I.P.aspx, 2006.
Visited at 15-08-2006.

Appendix A: Trinidad Reference Architecture

The architectural overview of the Trinidad Reference Architecture is separately added.

Appendix B: Trinidad Open Source Strategy

The Trinidad open source strategy is separately added.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Trinidad

Architectural overview

Version control

Version	Date	Short description changes
0.1	March 09, 2006	Start Report
0.2	April 27, 2006	First draft by Jermaine Jong
0.3	May 09, 2006	Filled in blank chapters
0.4	May 29, 2006	Deleted “Management Summary” Chapter Added info from .Net seminar Added clear scope definition
0.5	August 15, 2006	Minor changes according to feedback
1.0	<date>	Definitive document

Name author(s): Jermaine Jong

Trinidad

Name author(s): Jermaine Jong

Company name:	Capgemini N.V.
Place:	Utrecht
Date:	August 15, 2006

Preface / Introduction

This document contains an overview of the architecture of the Trinidad Platform, and gives an introduction to the Trinidad Platform and its agile methodology. Furthermore this document gives an elaborate description of the reference architecture and the framework supporting the Trinidad Platform.

The goal of this document is to serve as a reference for the Trinidad Reference Architecture and the Havana Framework. Just like the Trinidad Platform this architectural overview is a dynamic document which will be updated and changed during the development of the platform.

This document doesn’t provide a complete description of the Trinidad platform but merely a detailed description of two important concepts within that platform namely, the Trinidad Reference Architecture and the Havana Framework. The remaining concepts within the Trinidad Platform are only mentioned briefly in order to give the reader a clear view of the platform and the position of the described concepts within that platform.

The intended audience for this document exists of developers and architects wanting to get more insight in the technical details of the Trinidad Reference architecture.

Table of Contents

1 Trinidad Platform

1.1 Trinidad

The number of challenges modern software development projects encounter is expanding quickly. Think about integration with existing back-ends and third parties, service orientation, new platforms and media, and emerging technologies. For projects, executing on-time and on-budget becomes a diligent quest, with this ever increasing complexity. In order to deal with these challenges, Capgemini introduced the Trinidad Platform.

1.2 Components

The Trinidad Platform executes on the vision that projects need to be empowered and highly standardized in order to achieve high productivity and high quality at the same time. The Trinidad Platform consists of a number of integrated core elements.

- *Trinidad Lifecycle.* A flexible agile methodology, using best practices from Rational Unified Process (RUP), MSF Agile, extreme programming and Smart. An agile methodology is used to guarantee frequent, high quality delivery of working software. With this methodology comes a clear and easy-to-use estimation technique, based on pragmatic use cases, modeling guidelines and an online use case based planning and measurement tool. Furthermore, additional project and process support can be achieved by executing your projects in one of Capgemini's Accelerated Delivery Centers (ADC).
- *Trinidad Reference architecture.* Projects executed with the Trinidad Platform de facto apply platform independent multi-tier reference software architecture. Be it web, mobile or Windows development, employing databases, service oriented architectures (SOA) and enterprise busses. The reference architecture is supported by a broad and extensible framework, called the Havana Framework. This allows for maximum re-use of functional, technical built-in and third party services and components, such as authorization, SharePoint, web services, BizTalk, Microsoft Dynamics, and SAP. Capgemini gathered years of experience in constructing and using frameworks, such as Sculptor, CSLA and numerous project specific workbenches. The Havana Framework benefits from these experiences.
- *Model Driven Architecture.* The Trinidad Platform encompasses high quality code delivered at high speed. This is facilitated using model driven architecture (MDA). The flexible and extensible Tobago MDA Code Generator uses UML models from any UML modeling tool, and produces code according to predefined templates. Model driven architecture guarantees high quality and keeps testing effort low.
- *Havana Framework.* The Havana Framework is an easy extendable .Net framework, based upon well-known design patterns. Use of the Havana Framework speeds up development and increases the quality of software by offering common used, basic functionality

1.3 Trinidad Lifecycle

The Trinidad Lifecycle is the beating hart of every project.

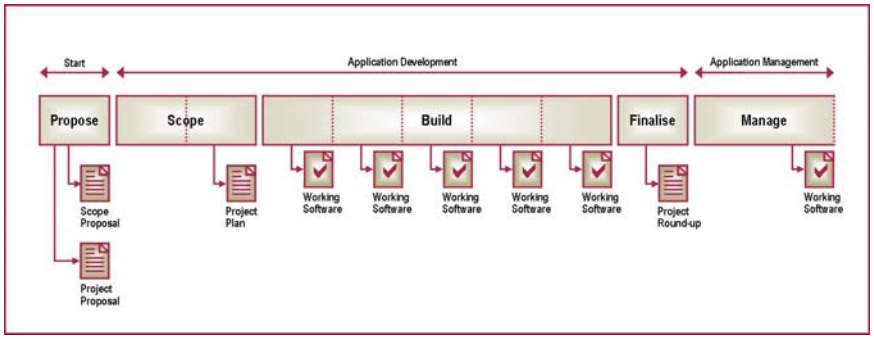

Fig. 1.1 Trinidad Lifecycle

The Trinidad Lifecycle exists of the following phases.

- *Propose.* The project's scope, size and complexity are determined roughly during a number of short intensive workshops. This stage leads to the initial project proposal.
- *Scope.* The project proposal is elaborated upon, again in workshops, leading to the plan of approach for the project. This stage includes use-case modeling and domain modeling, stakeholder and risk analysis, resourcing, and an estimate for the project.
- *Build.* The "Build" phase stands in light of interactively realising software. "Build" is divided into short iterations of preferably two weeks. During these iterations a number of smart use cases are realised. At the start of an iteration the use cases that will de realised are selected. This happens in the sub-phase "Plan". Next every iteration has a sub-phase "Build", in which the use cases are elaborated and realised. Deployment happens in the sub-phase "Run".
- *Finalise.* During the *Finalise* phase the project is finalized and evaluated.
- *Manage.* The (ongoing) stage *Manage* executes the maintenance of the delivered software. During this stage again smart use-cases underly the possible changes in the software. The stage *Manage* is usually executed in monthly or two-monthly iterations.

1.4 Tobago MDA Generator

The Tobago MDA Generator, Capgemini's flexible and extendable code generator produces a great deal of code in a Trinidad project according to predefined patterns and templates.

The use of model driven architecture and the Tobago code generator has a few important benefits. Development that is considered to be routine can be delivered at high speed, shifting the focus in the project to the business logic and complex functional and technical challenges. A gain in time is realized by maximizing reuse and capitalizing from experiences of preceding projects.

1.5 Havana Framework

The Havana Framework is a set of classes and other constructs that assist and speed up development for Trinidad based applications by eliminating repeating work. The use of the Havana Framework improves the quality of developed software. The Havana Framework is available for both .Net and Java.

The Havana Framework implements the layers and classes from the Trinidad Reference Architecture and handles the communication between those layers. The Havana Framework speeds up software development by alleviating routine work such as database connections and exception handling. Capgemini developed the Havana Framework for the following reasons:

- To create a common basis for all projects worldwide.
- To speed up development by facilitating reuse.
- To ensure a correct implementation of the Trinidad Reference Architecture.

1.6 Smart Estimator

Smart Estimator is a pragmatic use case based project estimation and measurement tool which provides dependable estimations that can be made in early stages of the project.

Nowadays most of the available use case based estimation techniques are coarse grained, therefore knowing a huge risk of deviation. The use cases are inappropriate as unit of work because they are not of equal granularity. Therefore Capgemini developed the Smart Estimator, which in fact supplies guidelines for modelling use cases and uses these use cases as unit of estimation. The use cases in Smart estimator are called Smart use cases.

1.6.1 Smart use cases

Smart use cases are from an equal small granularity and serve as suitable unit of work in the Trinidad Project Lifecycle. Smart use cases facilitate the re-use of requirements and allow for traceability in code. The unit of measuring complexity in the Smart Estimator is called a Smart use case point (SUCP). A Smart use case point is equal to approximately 1,4 function points.

Below are a few pointers on how to model use case diagrams:

- model actors and use cases
- keep use cases independent, use pre and post conditions
- keep use cases pragmatic, one form per use case

Below are a few pointers on how to describe use cases:

- use a pragmatic simple template
- describe the use case goal
- describe pre and post conditions
- describe the use cases steps

1.7 Trinidad Dashboard

The Trinidad Dashboard is an online project dashboard which clearly visualizes the scope and progress of the project. The dashboard helps stakeholders to monitor and manage a project and can assist in gathering metrics and decide on project velocity. The Trinidad Dashboard visualizes the progress in realizing use cases, as they are the unit of work in Trinidad projects. The dashboard also includes the possibility to create simple reports for different stakeholders, including management.

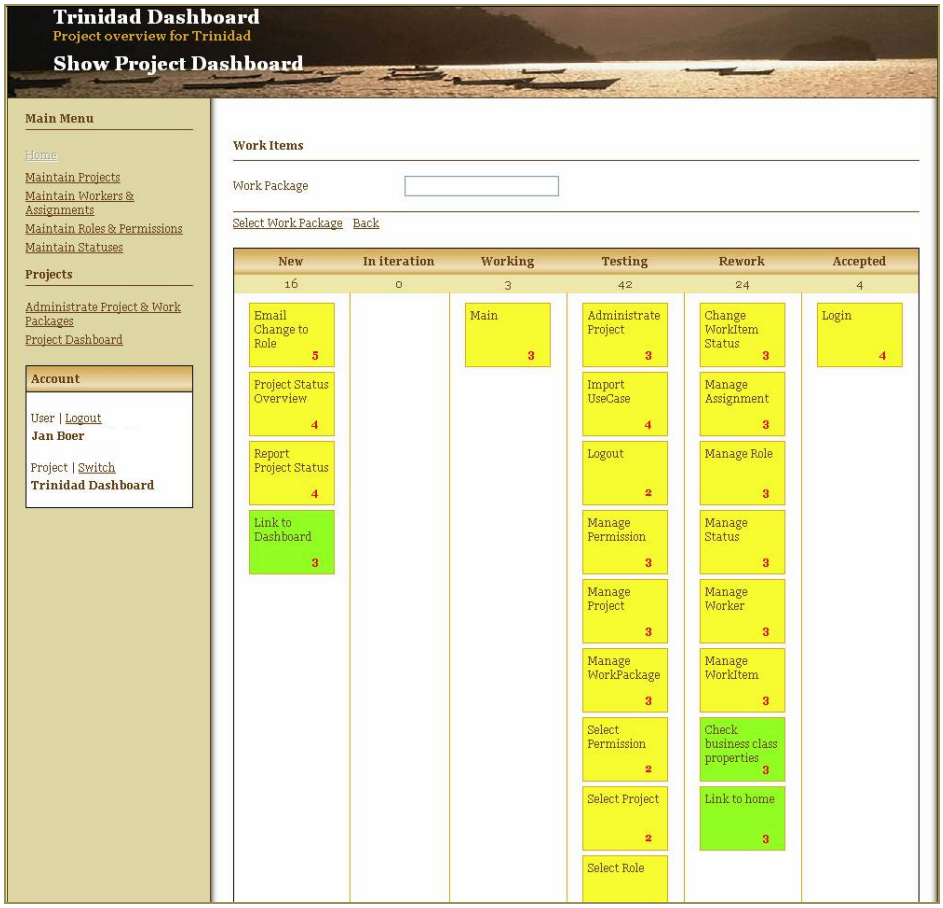

Fig. 1.2 Trinidad Dashboard

1.8 Benefits

Using the Trinidad Platform in software development projects realizes a few important benefits.

- *High quality*. Standardized guidelines for modelling and coding, optimal code generation and integrated tests lead to high-quality design and code.
- *High productivity.* Projects can be delivered on time and within budget because of the Trinidad Lifecycle and the optimal use of modelling and code generation.

- *Less testing.* The use of short agile iterations maximizes costumer and end-user's feedback. This rapidly increases the quality of the product and reduces the required testing effort.
- *Little and easy maintenance.* Because of the standardized approach, supported by the reference architecture requirements can literally be traced back to design and code. This speeds up maintenance.

2 Trinidad Reference Architecture

2.1 Goals

The Trinidad Reference Architecture is an important link in the realisation of applications and the stimulation of reuse. The reference architecture supports easy maintenance of produced software, because it guarantees traceability of requirements to code and design.

2.2 Overview

Because “reinventing the wheel” costs unnecessary time and money, Capgemini created the Trinidad Reference Architecture, one of the accelerators included in the Trinidad Platform.

The Trinidad Reference Architecture is a layered architecture. The use of layers is a good way to distinguish between the different responsibilities of the architecture, which accounts for a clear software architecture. The Trinidad Reference Architecture is created because of the following main reasons.

- The reference architecture presents a clear vision on how to develop software and introduces standardization
- The reference architecture assures the quality of software being built under the architecture because the architecture contains fully tested and proven components
- The reference architecture stimulates reuse over projects.
- The reference architecture diminishes architectural work in projects, only the deviation to the reference architecture needs to be explored.

2.3 Layers

The following figure shows an overview of the Trinidad Reference Architecture. The reference architecture has four layers which all have different responsibilities. Besides these four layers there is also a layer called the outside world. The layers and their functionality are further explained in the paragraphs below.

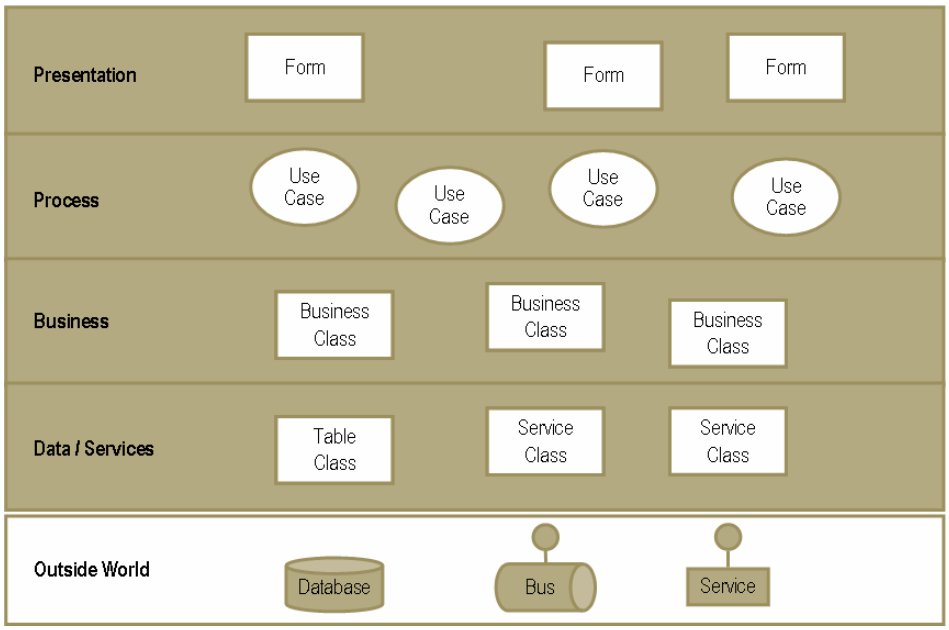

Fig. 2.1 Layered architecture

2.3.1 Presentation layer

The presentation layer presents information to the customer in a windows, web or even a mobile device application. Preferably, the presentation layer is a thin client which contains no logic, such as business logic or process logic, from other layers.

2.3.2 Process layer

The process layer is responsible for executing the use cases in the application. Every use case in the model is represented as a task, and the process layer manages these tasks. The process layer separates these processes from the rest of the application, making them flexible.

Use cases

The problem with most development techniques working with use cases is that the use cases often have a different level of granularity. The Trinidad platform uses smart use cases, which are from equal granularity.

During a Trinidad project a use case diagram is made for every elementary business process. In most cases this diagram represents a single user task, and consists of sub-function level use cases. Use case diagrams are used to reduce the complexity of functional requirements.

2.3.3 Business layer

The Business layer is the most important layer in the reference architecture. This layer contains all the business logic which is offered to the process layer. The business classes are located in the business layer. The business classes are derived from the classes of the domain model of the application. Every business object in the business layer is represented by a class in the domain model.

2.3.4 Data / Services layer

The Data / Services layer isolates the connections with data providers from the business layer. This layer provides the internal status of the business objects in the business layer. For every Business class in the Business Layer there's a Table or Service class in the Data / Services layer.

2.3.5 Outside World

The Outside World is everything in the surroundings of the application that is somehow connected with or has influence on the application. This "layer" represents the interfacing between the application and the outside world. Examples of components of the outside world are Siebel, CRM, SAP, Web Services, Databases, etc.

2.4 Blends

The Trinidad reference architecture can be implemented in a couple of blends which are described below.

2.4.1 Database

The Database variation is the standard reference architecture of the Trinidad Platform. This architecture represents a standard Trinidad application which consists of a webbased, Windows or mobile Microsoft .Net application, supported by an SQL server.

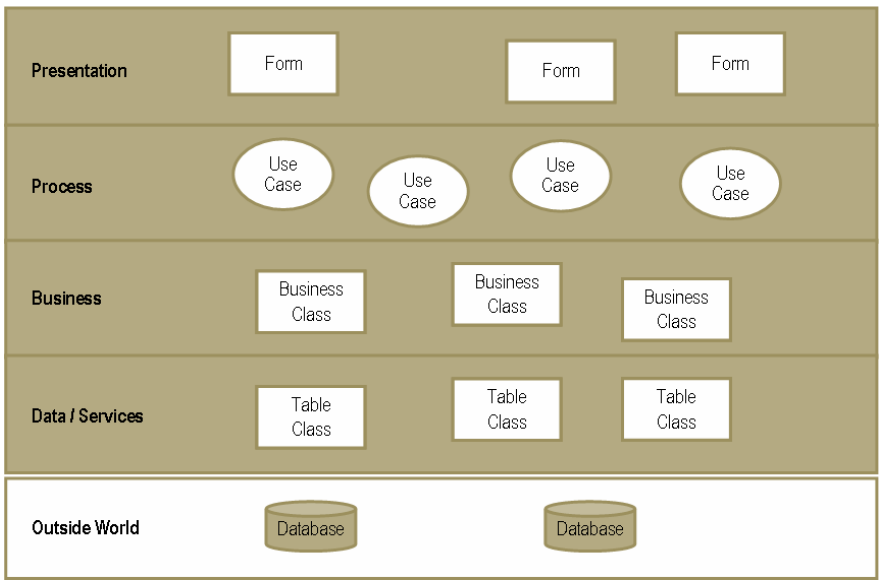

Fig. 2.2 Standard reference architecture

2.4.2 Service Consumer

Service Oriented Architectures are a growing trend. The Trinidad platform fully supports service oriented architectures. Fig. 2.3 shows a reference architecture which uses a service as data source.

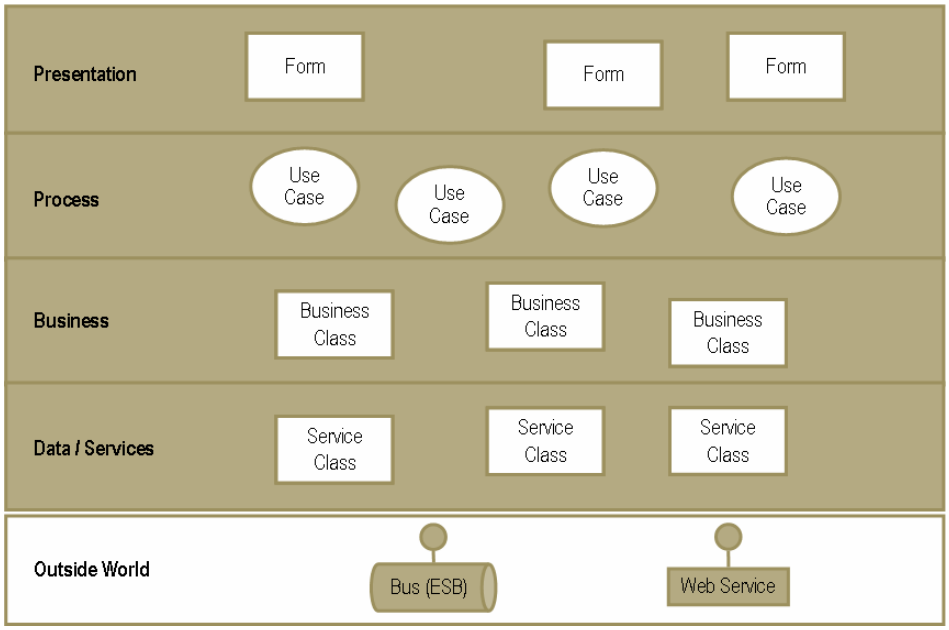

Fig. 2.3 Service consumer

As the outside worlds in Fig. 2.3 shows, the application can access the web service and use it as data source. The application can also easily access a bus structure which gets data from any other source which could be another web service, application, etc.

The only layer that has been modified in order to create this services client architecture is the Data / Services layer. Instead of Table classes this layer now contains Service classes in order to communicate with the service. Because of the layered structure of the architecture the other layers continue to function as if they were still using data directly from a database server. This example architecture contains multiple tiers; the application, the web service and the application delivering the web service.

2.4.3 Service Provider

The previous example showed a reference architecture which uses a service as the data source. What if we don't want to create an application which uses a service, but an application which itself functions as a service. This is also possible within the Trinidad Platform. An example service architecture could look like Fig. 2.4.

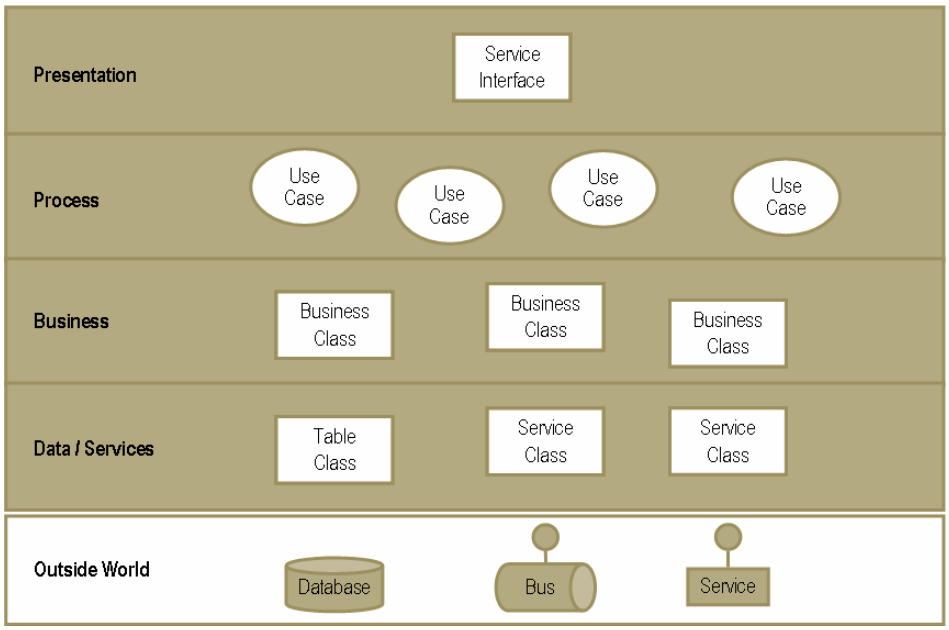

Fig. 2.4 Service provider

Looking at this figure we can see that the process and business layer are still identical to the standard reference architecture. In order to create a service the only layer that has to be changed is the presentation layer, instead of creating a form to represent the data a service interface has to be created. Again we can identify multiple tiers in this example architecture

The created service architecture can access data from a database server, but also from another data source. Fig. 2.4 shows the different possible data sources in the Outside world en Data / Services layer. Because of the layered architecture all kinds of variations are possible.

Fig. 2.5 gives a visual representation of some of the possibilities of applications in the Trinidad Platform. This picture shows a clear overview of how a service oriented architecture could be realised within the Trinidad Platform. The basis for every application is still the same, all these applications have more commonalities than differences.

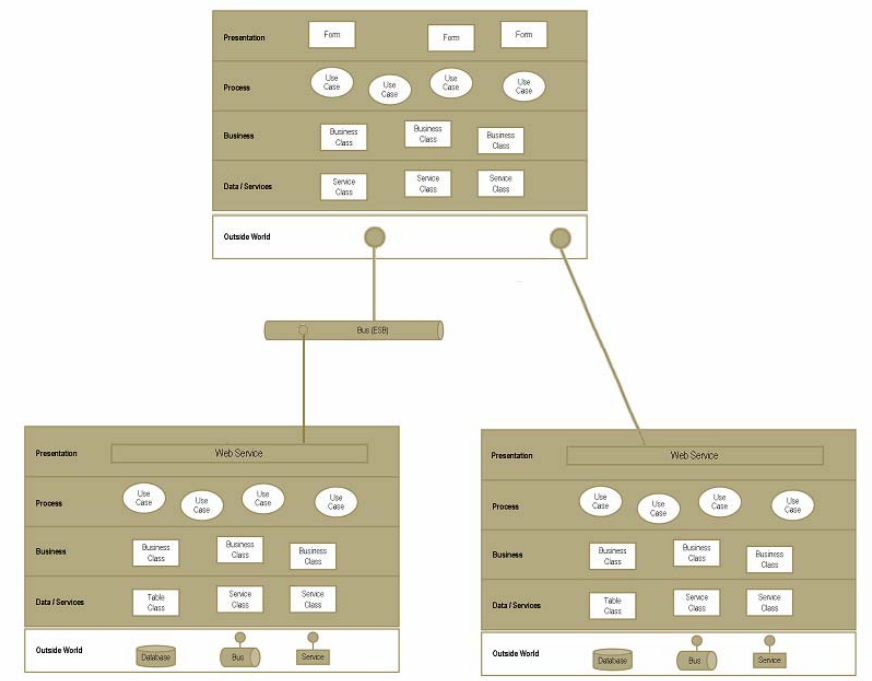

Fig. 2.5 Service Oriented Architecture

3 Framework-wide Concepts

The Trinidad Reference Architecture consists of a lot of components. Some of these components have a function through all the layers of the architecture and others aid development only in a particular layer.

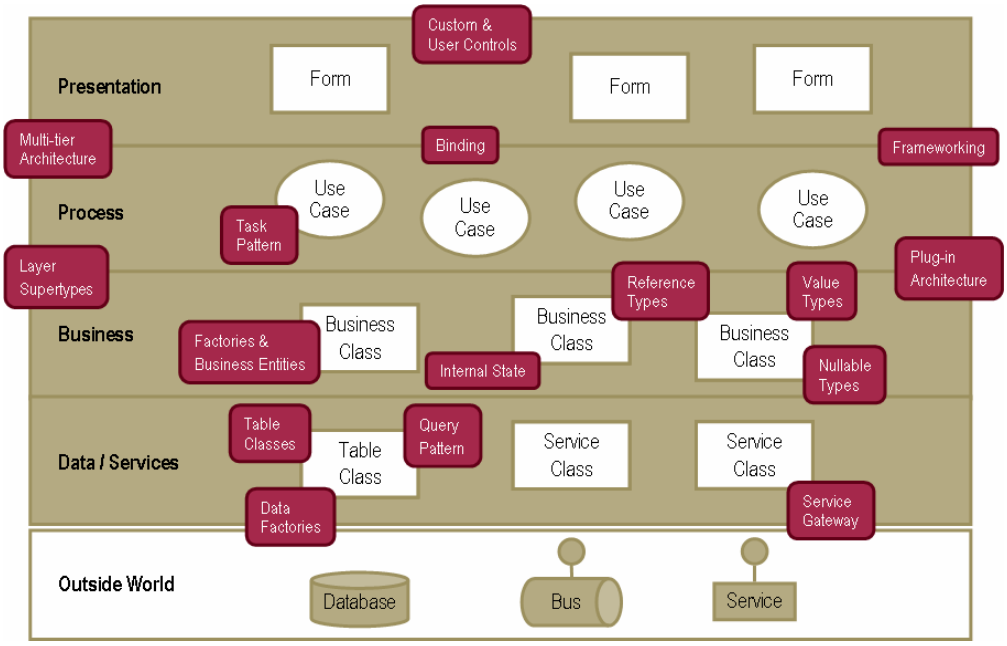

Fig. 3.1 Important components in the Havana Framework

Fig. 3.1 shows the Trinidad Reference Architecture complemented by the most important components. This picture also gives an overview of the specific components and the layers they are used in. Some of the components in Fig. 3.1 operate in a specific layer while others work through all the layers. The components discussed in this chapter are framework-wide components.

3.1 Plug-in Architecture

Nowadays applications can make use of all kinds of external services to fulfill a task. Depending on the wishes of the client applications need to support these services. Integrating with or switching from different services can sometimes be difficult. Applications constructed with the Trinidad Platform are built upon an extensibility mechanism that supports easy integration of external services. This mechanism is called the Plug-in Architecture.

3.1.1 Intent

Applications created with the Havana Framework are created upon a plug-in architecture. The framework is flexible and scalable enough to offer support for the integration of new components and external services.

A Trinidad based application gets its components and or services “plugged in” at startup. The component for lets say an authorization mechanism based on data from a database is added at startup. If the customer decides that the application

needs to use Active Directory authorization instead, the previous authorization mechanism is removed and the Active Directory authorization is plugged in. while the remainder of the application remains unmodified.

The plug-in architecture enables a great deal of flexibility in Trinidad based applications. All the required components can just be plugged into the application without further modifications.

3.1.2 How

The plug-in architecture is based on the Bridge pattern. The Bridge pattern is intended to decouple an abstraction from its implementation so that the two can vary independently. The use of this pattern enables the easy implementation of external services in Trinidad based applications.

The Bridge pattern is useful when the implementation must be exchangeable (during runtime). The abstraction and implementation in the Bridge pattern can both be extended by deriving subclasses from them. Changes to the implementation have no effect on the abstraction or the clients using the abstraction. The client code doesn't even have to be compiled again which enables runtime switching between implementations.

Fig. 3.2 shows the standard view of the Bridge Pattern in UML. The Bridge pattern exists of the following classes or objects:

- *Abstraction*, Defines the abstraction's interface and maintains a reference to an object of type Implementor.
- *RefinedAbstraction*, Extends the interface defined by Abstraction.
- *Implementor*, Defines the interface for implementation classes. This interface doesn't have to correspond exactly to Abstraction's interface; in fact the two interfaces can be quite different. Typically the Implementation interface provides only primitive operations, and Abstraction defines higher-level operations based on these primitives.
- *ConcreteImplementor*, Implements the Implementor interface and defines its concrete implementation.

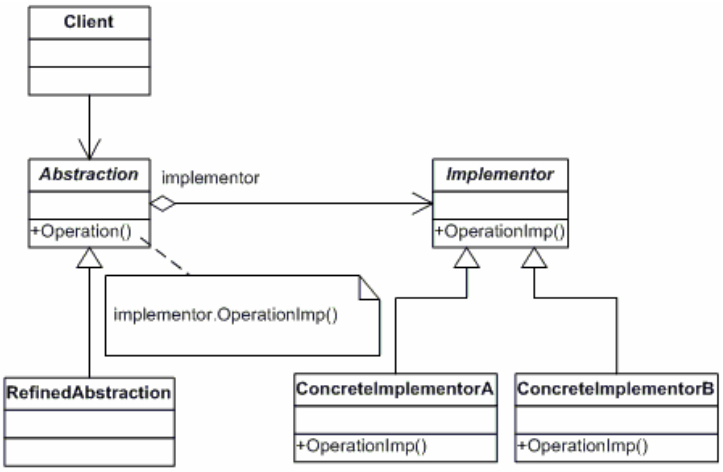

Fig. 3.2 Bridge Pattern

Every type of pluggable component in a Trinidad application contains a plug-in manager class (the abstraction) and an interface (the implementor). Continuing with the previous example, a Trinidad application would contain an Authorizationmanager and an IAuthorizer interface. All the authorization mechanisms in the Framework implement the IAuthorizer interface.

Using the plug-in architecture in a Trinidad application makes it easy to add, remove and switch from different implementations. At startup the implementations are plugged into the abstractions. When the implementation is used the client calls the abstraction, which routes the call to the implementation. The implementations to use are defined in the startup script of the application. Figure 4.3 shows a code example that defines the implementations that will be plugged into the application at startup.

```
Central.Init();
Central.State = new WebState();
Central.Resource = new DatabaseResourceManager();
Central.Navigator.NavigationFactory = new NavigationFactory();
Central.ExceptionHandler = new WebExceptionHandler();

Binder.Init();
Binder.AddControlBinder(new DataGridPager());
Binder.AddControlBinder(new DataGridSorter());

Binder.AddBusinessBinder(new BusinessBinder());
Binder.AddBusinessPersister(new BusinessPersister());

Binder.AddDataBinder(new DataGridBinder());
```

Fig. 3.3 Definitions for a Web based application.

The code example in Fig. 3.3 origins from a Webbased application. The plugged in implementations that are defined in this code example are specifically designed for a webbased application. The state of the application is controlled by an implementation called, the WebState, the exceptions are handled by the WebExceptionHandler, etc.

The application in Fig. 3.3 can easily be adapted to a windows application. The specific webbased implementations are just plugged out of the application while windows based implementations are plugged in instead.

```
Central.Init();
Central.State = new WinState();
Central.Resource = new ConfigResourceHandler();
Central.Navigator.NavigationFactory = new NavigationFactory();
Central.ExceptionHandler = new MessageBoxExceptionHandler();

Binder.Init();
Binder.AddBusinessBinder(new BusinessBinder());
Binder.AddDataBinder(new DataGridBinder());
```

Fig. 3.4 Definitions for a Windows based application

The actual code that makes use of these implementations doesn't need to be changed or modified. Fig. 3.4 shows the windows implementations of the application. The WebState is replaced by a WinState, the resource management differs, etc.

Additionally it is possible to define an array of implementations for a specific component. If an application has e.g. different kind of logging mechanisms they can both be used in the application by inserting them in an array and defining that array as implementation for the logging control. This is known as the so called "multiple Bridge pattern".

3.1.3 Benefits

The plug-in architecture has a great impact on the total reference architecture, and is frequently used throughout the whole architecture. The plug-in architecture makes the reference architecture flexible and scalable. Stated below are the most important benefits that the plug-in architecture offers.

- A clear separation of definition and implementation.
- Replaceable and exchangeable omponents.
- Allowing easy integration in all kinds of external services such as user interface, databases, web services, enterprise service busses.

3.1.4 Consequences

There are a few consequences of using the plug-in architecture, some of which have already been mentioned at the benefits of the plug-in architecture. The consequences of using the plug-in architecture are stated below.

- *Decoupling interface and implementation,* An implementation is not bound permanently to an interface. The implementation of an abstraction can be configured at run-time. It's even possible for an object to change its implementation at run-time.
- *Improved extensibility,* You can extend the Abstraction and Implementor hierarchies independently.
- *Hiding implementation details from clients,* You can shield clients from implementation details, like the sharing of implementor objects and the accompanying reference count mechanism (if any).

3.2 Layer Supertypes

A layer supertype is an object who acts as a supertype for all the types in its layer. All the actual types in that layer inherit from the layer supertype. Every layer in the Trinidad Reference Architecture contains layer supertypes.

3.2.1 Intent

Layer supertypes force common features on inherited types. The use of layer supertypes prevents redundant code. If we look at the business layer for example, we see that this layer contains a lot of business classes (one for every type of business object). These business classes share a lot of common functionality. Instead of recoding this shared functionality into every single business class, it is raised a level of abstraction higher and put in a layer supertype. Every business class in the business layer now inherits from the layer supertype.

3.2.2 How

Every layer supertype is actually an abstract class from which other classes in the layer can inherit. The layer supertype is initially empty, but will be supplemented with common functionality during development. Fig. 3.5 shows a few layer supertypes in a project. The BusinessClass in the Business package is the layer supertype for every business class in the business layer.

Every business class implements the IBusinessClass interface. Calls from anywhere in the application are made to objects of the type IBusinessClass. The benefit of this is that external third party business object can simply implement this interface in order to be recognized as a business object by the rest of the application.

Fig. 3.5 Layer Supertypes

Another layer supertype is the Table Class in the Data / Services layer. Fig. 3.6 shows a code snippet from the TableClass layer supertype.

```
public class TableClass
{
  protected const string All = "*";
  protected const string CountAll = "Count(*)";
  public const string ID = "ID";

  public static TableClass Empty = new TableClass();

  public static DataState New(Databases database, string table){ … }
  protected static DataState Get(Databases database, string table, ID id) { … }
  public static bool Save(Databases database, DataState state){ … }
  public static bool Delete(Databases database, string table, ID id) { … }
  protected static DataState GetAll(Databases database, string table){ … }
}
```

Fig. 3.6 TableClass layer supertypes

3.2.3 Benefits

Layer-supertypes are good object oriented programming practices. Layer supertypes prevent redundant code by abstracting common functionality to a higher level. Using layer supertypes also supports maintainability of the application. Changes in the way a table class handles operations on data, for example a conversion between data types, can be made only in the layer supertype.

3.2.4 Consequences

- *Prevention of redundant code*, The application will not contain any redundant code because all the common functionality is placed in the layer supertype.
- *Carefully select supertype data*, Realize that not everything can be placed into the supertype, object specific code should be in the object itself instead of in the layer supertype.
- *Maintainable code*, Changes can be made to a central class, the layer supertype.

3.3 Frameworking

If you want to speed up development and eliminate rework you can't go without frameworking in a reference architecture. A Framework is a set of classes and other constructs that assist and speed up development. The Trinidad platform uses the Havana Framework.

3.3.1 Intent

The main purpose of a framework is to find a solution to a problem in a given context that is originally used in a different context, in other words: reuse. A

framework can become a solid base for building applications, containing fully tested, high quality code.

3.3.2 How

While creating a framework the target architecture must be kept in mind. There are different strategies to create a framework. Frameworks can be implemented using design patterns, or even be based on an open source framework.

Below are 2 strategies to create a framework.

- *Up-front*, Creating the entire framework before building the application. Working according to this strategy carries the risk of creating unnecessary items. It is difficult to foresee the required functionality, therefore difficult to plan.
- *Dynamic*, This is the opposite of the up-front strategy, Code that is needed in the framework is created when it is needed, just-in-time. Using the Layer Supertype pattern is a good example of dynamic framework building.

The Dynamic strategy seems like the wiser strategy to choose while setting up a framework. Just create what is needed, when it is needed and add it to the Framework.

3.3.3 Benefits

Besides speeding up development by eliminating rework, the use of a framework also stands for an improved quality of the code. The code in a framework has been used and re-used many times before. Therefore this code is fully tested and not likely to contain an error.

3.3.4 Consequences

An essential part of building up a good framework is to only implement code in the framework that has already been used. A framework needs to be a collection of proven code and patterns. Harvest and refactor in order to create a structured framework. Layer Supertypes, Plug-in Architecture and Refactoring tools such as Resharper support this process.

3.4 Binding & Persisting

The Trinidad reference architecture is a layered architecture, with every layer carrying his own responsibility. Although these layers operate independently they have to share data with each other somehow. This is done through the process of binding and persisting.

3.4.1 Intent

The binding and persisting mechanism is actually an in memory representation of data contained in a record set, and is used to pass data around. This can be data from databases, tables and rows but also data from other sources such as XML and web services. This data representation can easily be connected to presentation widgets.

Instead of having every layer making his own call to the data source the data is accessed in a single layer and passed around through the binding and persisting mechanism. This brings structure to the dataflow, and limits the data calls.

3.4.2 How

As the name already reveals, the binding and persisting process exists of two separate activities, namely binding and persisting.

When a user requests information through e.g. the user interface the binding process is started. The requested data is gathered and presented to the user. The left arrows in Fig. 3.7 show the flow of the binding process.

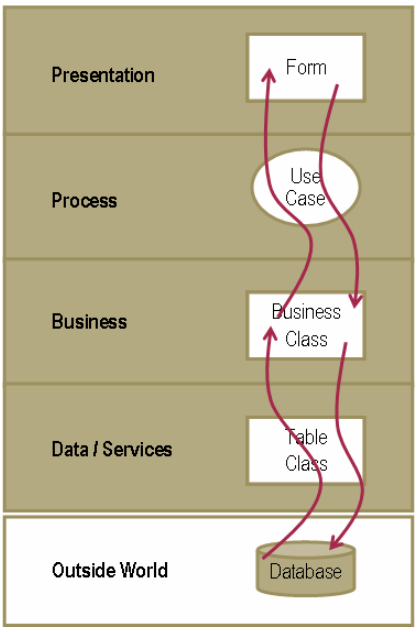

Fig. 3.7 Binding and Persisting

When data is committed through e.g. a webform in the user interface, the persisting process takes place. The data in the user interface is saved to its source. The right arrows in Fig. 3.7 represent this process.

As we can see in Fig. 3.7 both of the processes have an interruption in the business layer. Trinidad applications bind to business classes instead of datasets. The business layer is in full control of the data and contains a Bindmanager. Upon a data request the Bindmanager receives the object and the business class to which to bind. The business class is searched by means of reflection and the properties of the data object are bound to the properties of the business class. However, the property names of the two objects must match in order to bind them together.

The business layer delivers the right data and decides on correctness of changes made to the data.

3.4.3 Benefits

The Binding & Persisting mechanism localizes data calls and realizes a structured data-flow.

3.4.4 Consequences

The major consequence of using the binding and persisting mechanism is that the Business layer has got to have full control of the data. The Business layer decides on correctness of changes made to the data.

In order for the binding and persisting process to work all data requests should go through the business layer, record sets are not to appear above the business layer. The application must also be data source independent.

3.4.5 Alternatives

The Microsoft data binding architecture (Mdba) is an alternative for the Binding and persisting mechanism used in the Trinidad Platform, the Smarter binding architecture (Sba). These two mechanisms differ from each other on the following points:

- The Mdba automatically binds properties of controls to elements in the data source while Sba binds to business classes instead of DataSets and uses reflection to locate bindable properties. Sba can create data bindings on the fly which increases the flexibility and makes the mechanism more dynamic.
- The Mdba has different windows and web implementations while Sba is transparent to windows or web.

4 Presentation

4.1 Custom & User Controls

The user interface is an important part of an application. The interface forms the view of the application to the outside world and consists of forms and controls, which are used to operate the application. Trinidad based applications are built on user and custom controls located in the framework.

User controls are designed visually in Visual Studio, by dragging and dropping them while custom controls are designed programmatically.

4.1.1 Intent

Custom and user controls are specific user elements, designed to fulfill a specific function. Custom controls are mostly standard controls with added functionality which are designed to simplify the most recurring actions in applications.

4.1.2 How

User controls are the standard controls which are available in Visual Studio. These controls can be dragged and dropped on top of a page. The properties of these controls can be modified using the property explorer, and double clicking a user controls reveals the code behind them.

Custom controls are programmatically designed controls that serve a specific function in a Trinidad application. An example of a custom control is “NumericText”. This control only accepts numeric values and is an adapted version of the default .NET textfield control. This control inherits from the .Net control class. Fig. 4.1 shows how properties like the style sheet class or the control’s state can be customized.

```
public class NumericText : Control, IPostBackDataHandler
{
   public NumericText()
   {
            CssClass = "";
            Text = "";
            Enabled = true;
   }

   public string CssClass
   {
            get{ return (string) ViewState["CssClass"]; }
            set{ ViewState["CssClass"] = value; }
   }

   public bool Enabled
   {
            get{ return (bool) ViewState["Enabled"]; }
            set{ ViewState["Enabled"] = value; }
   }
...
```

Fig. 4.1 NumericText

4.1.3 Benefits

User controls are easy to use. They can by easily added to a page by dragging and dropping them and contain easy event handling.

Custom controls simplify the development of the most recurring actions in applications and can be even be generated automatically.

Both of these controls result in a richer user interface.

4.1.4 Consequences

Custom controls contain a certain amount of business logic, they can for example check on the correctness of input. There needs to be a clear separation between the type of business logic in the business layer and the controls. The designer must be aware of this and make explicit choices on where to place certain business logic.

5 Process

5.1 Task Pattern

Pragmatic use cases (smart use cases) are the unit of work in Trinidad projects. The use cases are implemented in the Process layer, and managed by the Task pattern. The Task pattern states that each use case should be implemented as a single separate task and handles the navigation between tasks as well.

5.1.1 Intent

Use cases are the de facto standard for modeling functional requirements. Implementing use cases as separate tasks allows for traceability in code, and re-use of use cases. The task pattern is a mechanism that provides a way to implement these use cases separately, and to communicate and navigate between the use cases.

5.1.2 How

Every use case is implemented as a separate task. Each task that is implemented inherits from the base class called Task. The base Task class is a layer supertype for tasks. This layer supertype contains, among other methods, a validation and execution method (Fig. 5.1).

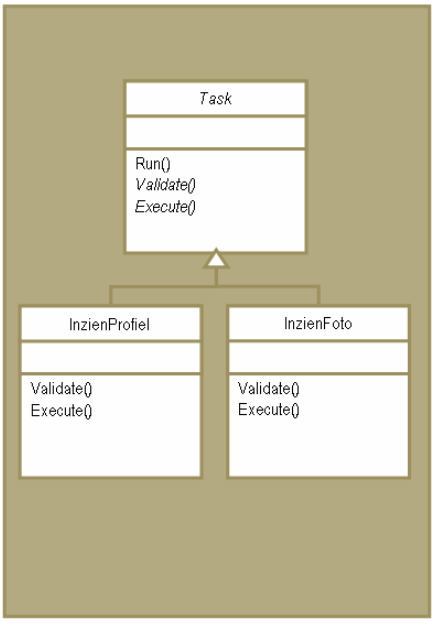

Fig. 5.1 Task Layer Supertype

Every use case contains pre-conditions. When a use case is started the task validates its pre conditions before executing the use case steps. After successful validation the task handles the use case actions in order to realize the use case goals and make the post conditions come true. Use case actions can vary from starting a user interface screen to even starting another use case.

In order to keep the tasks independent a “Router” is implemented in the process layer. The Router functions as a communication mechanism in the process layer. The tasks only communicate with this router. As shown in Fig. 5.2, a task communicates with another task through the router.

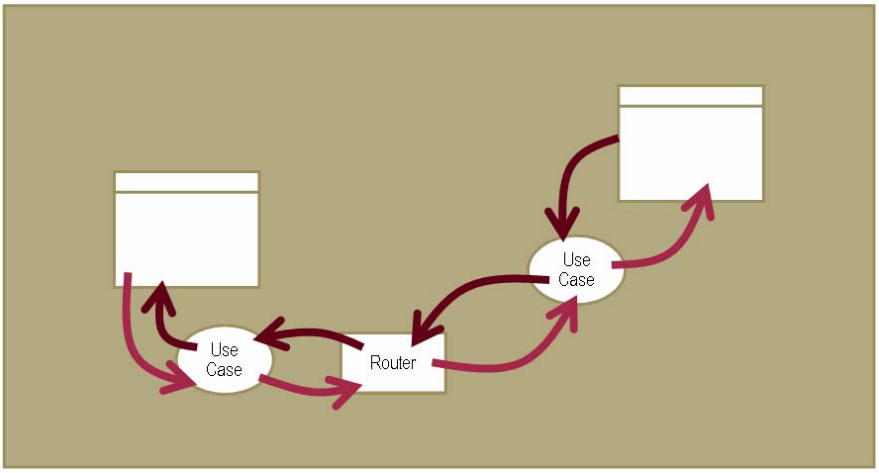

Fig. 5.2 Task Router

The communication between tasks and the principal of the router are based upon the Command pattern.

5.1.3 Benefits

Using the Task pattern enables the re-use of use cases. If any other task requires functionality from another previously implemented use case the task can simply go back and connect to the use case via the router. An additional benefit is that the use cases can be used as unit of work.

5.1.4 Consequences

Pragmatic use cases must be used in order to implement every use case as an independent task. The use cases must be independent and have to contain specific pre and post conditions.

Every use case needs to be implemented as its own task. The use of the task pattern supports incremental development as defined in the Trinidad life cycle.

6 Business

6.1 Factories & Business Classes

The primary function of the business layer is managing the business logic of the application. Trinidad applications use factories and business entities as way of dealing with business logic.

6.1.1 Intent

The intent of Factories and Business classes is to manage the business logic, and providing a mechanism to link the business classes to the database.

6.1.2 How

Modeling business entities is essential. Before implementing the business entities a domain model which contains all the business entities is modeled. The domain model (Fig. 6.1) contains the properties, the property types and the relationships between the business entities. Each class represents an entity in the real world.

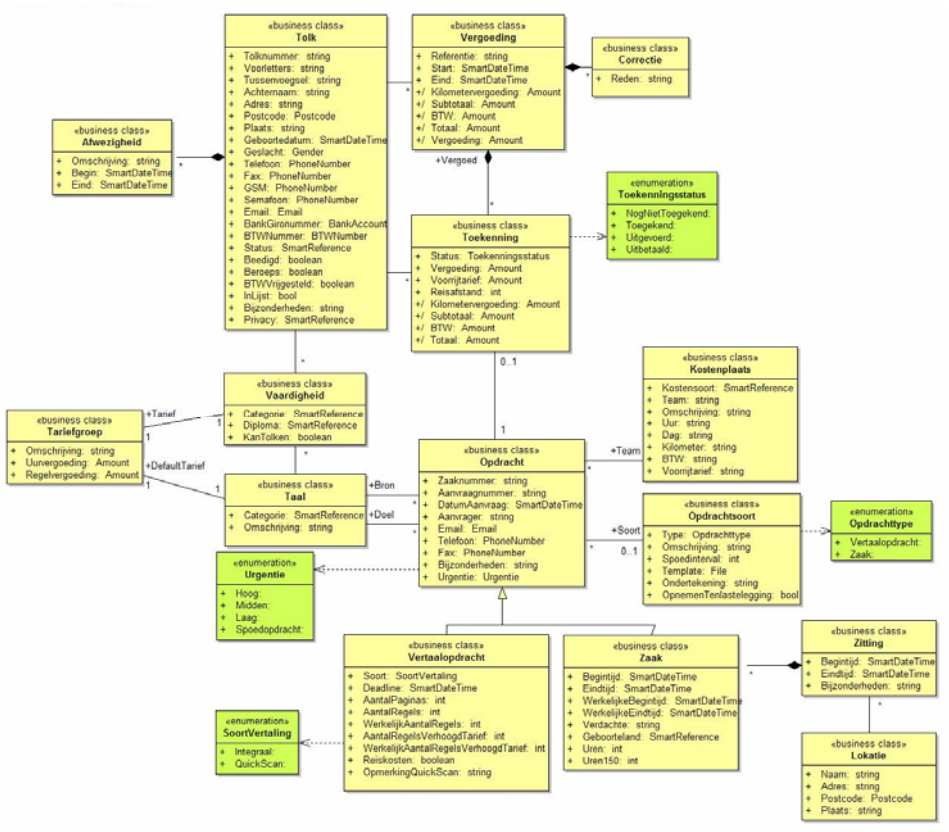

Fig. 6.1 Domain model

A Factory is an object that creates and possibly manages other objects. Within the Trinidad Platform the Factory is responsible for executing business services and managing the life-cycle for business entities.

A business entity is instantiated by a factory. A business entity contains business logic on a single instance and implements the internal state (§ 6.2) as shown in Fig. 6.2.

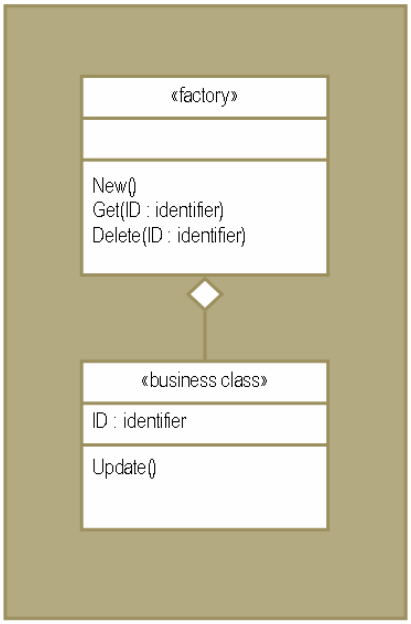

Fig. 6.2 Business factory

6.1.3 Benefits

The use of a Factory to create business objects keeps the code reusable and extensible. The business objects assure data-hiding, the objects only "know" about the data they need in order to fulfill their task. This reduces the possibility that wrong data is changed inadvertently during maintenance.

6.2 Internal State

The business entities in the business layer are stateless, they are objects containing the business logic of the corresponding business entity. The actual state of a business entity is encapsulated by the Internal State. The business entity is unaware of the way the state is retrieved and persisted.

6.2.1 Intent

The business objects in the business layer can have a number of properties. Instead of keeping the value of these properties in the business object itself, these values are provided by the internal state that every business object contains. This provides a clean separation of concerns, the business object only contains the business logic while the internal state handles the state of the object’s properties

6.2.2 How

The implementation of the Internal State is based upon the State Design Pattern. Every Business class has an internal state object which implements an IInternalState interface. Through this interface the Business class gets and sets the actual data (Fig. 6.3).

```
public class Customer : BusinessEntity
{
    public static Customer New() {…}

    public static Customer Get(ID id)
    {
          state = TableCustomer.Get(id);
    }

    public bool Save()
    {
          return state.Save();
    }

    public string FirstName
    {
          get { return state.Get(CustomerProperties.FirstName); }
          set { state.Set(CustomerProperties.FirstName, value); }
    }

    public Contact Manager
    {
          get { return state.GetEntity(CustomerProperties.Manager) as Contact; }
          set { state.Set(CustomerProperties.Manager, value); }
    }
}
```

Fig. 6.3 Internal State

For every data source there is a different state object. Fig. 6.3 shows a graphical overview of the Internal State. Every state object implements the interface IInternalState. DataState is responsible for getting and setting data in a database. DataState internally stores the data in a DataSet.

Another possible implementation of the Internal State is to create a Layer Supertype which implements the IInternalState interface. In Fig. 6.4 the XMLState and the WebState implement from the Layer Supertype InternalState. Both of these states communicate with a different data source.

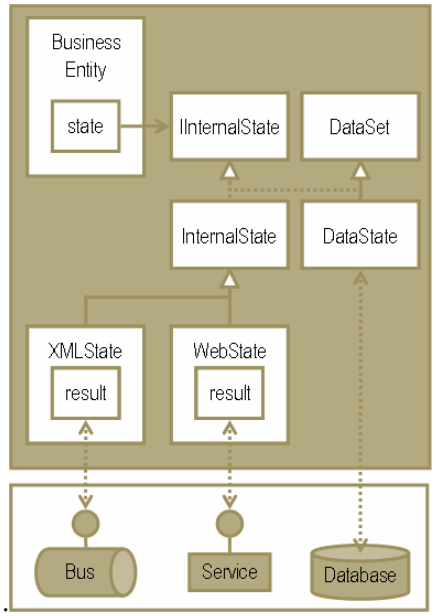

Fig. 6.4 Internal State overview

6.2.3 Benefits

The biggest benefit of the Internal State is that the business entities are unaware of the way data is acquired or the source the data origins from. This supports the data source independency of the business layer.

6.2.4 Consequences

The Internal State manages the status of an object's data and allows the data to be changed without the business entity being aware of where the data comes from. In order to support additional data sources, internal states have to be made to handle every specific data source.

6.3 Reference Types

The business logic of most application deals with collections. These collections can vary from a list of countries from which to choose, to different states of a product. At first glimpse, enumerated types and referential classes appear to be to the most obvious patterns for implementing reference types, but there are other patterns that can do the job for you. The Havana Framework contains five patterns for dealing with reference types:

- Enumeration
- String collection
- Descriptor
- Small business class
- Smart Reference

6.3.1 Intent

Every pattern that can be used for dealing with reference types has advantages and disadvantages. The framework contains these five patterns in order to keep the platform flexible and to assure a suitable pattern for every situation.

6.3.2 Enumeration

When you have a fixed number of elements to choose from, an enumeration is a suitable pattern for implementing references. An enumeration is a hard-coded collection of values. The enumeration Levels in Fig. 6.5is an excellent example.

```
public enum Level
{
        Low,
        Medium,
        High
}
```

Fig. 6.5 Enumeration

If somehow the value of an element needs to be changed or new elements need to be introduced, enumerations lack the flexibility to perform this operation without recompiling the application. Enumerations are also implemented as a value type in .NET, which means that there is no way of extending them.

6.3.3 String collection

In this pattern the collection of possible values is implemented as a class with a number of constants defined as its fields, like in Fig. 6.6. Such a class can easily be inherited from.

```
public class Level
{
        public static const string Low = "Low";
        public static const string Medium = "Medium";
        public static const string High = "High";
}
```

Fig. 6.6 String collection

One can only retrieve all possible values using reflection. This may not be very desirable. Even worse is that constant collections are not always type safe, even if they appear to be at first sight.

6.3.4 Descriptor

The third pattern, the descriptor pattern, mixes best practices from the previous two patterns. Basically, a descriptor is a collection of instances of the class itself. Using this pattern, the Levels class can be implemented as follows:

```
public class Level : Descriptor
{
        public static const Level Low = new Level("Low");
        public static const Level Medium = Level("Medium");
        public static const Level High = Level("High");
}
```

Fig. 6.7 Descriptor

Because the instances are again static, descriptors are type safe, both in simple checks as in operation signatures, just like enumerations. Descriptors have a fixed number of elements, just like enumerations and constant collections. And again, like the latter, retrieving the set of possible values requires reflection.

6.3.5 Small business class

Suppose you have a Location class that specifies a location where courses are held. Such a class would have a property Country that refers to a specific instance of a reference type Countries, such as in figure 7.8.

In this implementation of the Location class the property Country can be set to any valid instance of the Countries class. The Countries class is a small business class. New elements are hardly ever entered, existing elements rarely change and small business classes are mostly used as references. In most cases, the data of instances of small business classes are kept in a database, each small business class having an associated table in the database. Regardless of how the data is retrieved from the database, a small business class may look something like Fig. 6.8.

```
public class Location
{
   public string Name;
   public string Address;
   public string City;
   public Countries Country;
}
```

```
public class Countries
{
   public string Name;
   public string Description;
   public string CountryCode;
   …
   public static Countries New(){ … }
   public static Countries Get(ID id){ … }
   public bool Delete(){ … }
   public bool Update(){ … }
   public static Countries[] All
   {
      get { … }
   }
}
```

Fig. 6.8 Small Business Class

Using a small business classes as reference allows for great flexibility. That is, it is easy to change or add an element, without having to recompile the application. Another benefit of using small business classes is that retrieving the collection of all instances is straightforward. It is not unlikely that a small business class holds

a static property All (or likewise operation) that returns the collection of all countries, as an array of instances of Countries. And more, it is easy to inherit from a small business class and add additional functionality.

6.3.6 Smart Reference

In the last pattern, a single small business class is defined. This single small business class is used to refer to any reference type that can be expressed in the properties of that class. Typically such a class, called a smart reference, has properties such as Name and Description. The single-most important property of a smart reference however, is the Type property, which declares which specific reference type is meant.

The Type property can best be expressed using an enumeration, for instance called ReferenceTypes, as in the following code example. Using this property, operations such as All() and Default() can be defined for retrieving all instances of SmartReference for a specific type, or even the default instance. Such operations can be declared both static and not static, either specifying the type, or using the instance's Type property.

```
public enum ReferenceTypes
 {
    Countries,
    Currencies
 }
 public class SmartReference
 {
    public string Name;
    public string Description;
    public ReferenceTypes Type;
    …
    public SmartReference[] All() { … }

    public static SmartReference[] All(ReferenceTypes rt)
    { … }
 }
```

Fig. 6.9 Smart Reference

This last pattern is extremely flexible. New elements can easily be added to any of the types in the ReferenceTypes enumeration. The main benefit of using a smart reference over a number of small business classes is that it saves you a lot of programming effort. Maintenance functionality for all your reference types consists only of a single form, allowing the user to select a reference type and maintain the elements of that type.

6.3.7 Benefits

Collections are used in every application. Every reference type contains different characteristics and is applicable under different circumstances. Implementing a

variability of Reference types in the framework supports the flexibility of the Trinidad Platform.

6.3.8 Consequences

Using references is inevitable when using business classes. However, there a lot of different implementations, all of which are variations or extensions to one of the five patterns mentioned above. The big issue is when to use which pattern. Table 6.1 sums up the characteristics for each of the patterns.

	Type Safety	Collections of values	Display values	Flexibility	Extensibility (inheritance)
Enumeration	Yes	Yes	Limited to enumerated values	Fixed, change of code required	No
String collection	Not when used as parameter	Yes, using reflection	Limited to defined values	Fixed, change of code required	Yes
Descriptor	Yes	Yes, using reflection	Yes	Fixed, change of code required	Yes
Small business class	Yes, when defining static instances	Yes, easily retrieved from database	Any combination of its fields	Flexible, alter dedicated table	Yes
Smart reference	No, anonymous	Yes, easily retrieved from database	Any combination of its fields	Flexible, alter single table	Yes

Table 6.1 Characteristics

The first three patterns are best in situations where a limited and fixed number of elements are expected. The three patterns mainly differ in that different display values can be used or where extensibility by inheritance is required. The last two patterns are best used in situations where there is a large collection of elements, and this collection is likely to change more often than the code is put in production. This requires the elements to be stored externally, rather than being hard coded, most commonly in a database.

Together, these five patterns, and the numerous variations and extensions to them, will give you control over the references you need to deal with in your projects.

6.4 Value Types

Value types are predefined custom types, used to store data that has a certain standard format. Examples of possible value types could be an email address, a phone number, etc.

6.4.1 Intent

Define your own data types as value types is a good practice. A value type contains custom validation rules to ensure that the right format is stored in the value type. Value types also enable automatic casting.

6.4.2 How

All value types are derived implicitly from the Object class. Fig. 6.10 shows a few examples of value types.

```
public Email Email
{
  get { return state.Get(TableMedewerker.Email); }
  set { state.Set(TableMedewerker.Email, value); }
}

public PhoneNumber Telefoon
{
  get { return state.Get(TableMedewerker.Telefoon); }
  set { state.Set(TableMedewerker.Telefoon, value); }
}

public Postcode Postcode
{
  get { return state.Get(TableMedewerker.Postcode); }
  set { state.Set(TableMedewerker.Postcode, value); }
}
```

Fig. 6.10 Value types

6.4.3 Benefits

Value types ease the use of data in a specific format by eliminating the need to check on the data. Value types contain their own validation rules.

6.4.4 Consequences

Unlike reference types, it is not possible to derive a new type from a value type.

6.5 Nullable Types

Nullable types are regular base types like booleans and integers with the difference that they can be given a null value.

6.5.1 Intent

Nullable types can be useful to store information from which the value is not yet known at the moment. These could be values like date of birth, number of attendees in a course, etc.

6.5.2 How

Nullable types are custom made data types. These data types are sometimes based on regular data types complemented with additional functionality. Looking at Fig. 6.11 we can see that the NullDateTime object is based upon the regular DateTime object, but the NullDateTime contains extra functionality like IsNull.

```
NullDateTime dt = DateTime.Now;
NullDateTime dt2 = "2005-1-1";
NullDateTime dt3 = "";
NullDateTime dt4 = null;

if (dt2 == dt3) {…}
if (dt2.IsNull) {…}
```

```
int? x = 123;
int? y = null;

if (x.HasValue) Console.WriteLine(x.Value);
if (y.HasValue) Console.WriteLine(y.Value);
```

Fig. 6.11 Nullable Types

Nullable types are custom build in the Trinidad Platform, but are also supported by the .Net Framework 2.0.

6.5.3 Benefits

A value is not always known or can not be defined at initialization. With the aid of Nullable types these values can be assigned the null value, until they are needed or until their value is known.

6.5.4 Consequences

A data type can be both a Nullable and a Value type. By defining a null value for a Value type it can act as a Nullable type as well.

7 Data / Services

7.1 Table Classes

When the application uses a database as the data source, the Data / Services layer contains a Table Class for every business entity in the domain model. The Table Class communicates with the data source, ensuring that execution of database logic is isolated from the business entities.

7.1.1 Intent

Isolating communication with the data source to only the Data / Service layer supports database independence. The Table Classes provide this isolated communication. Because all the database specific code is located in the Table Class, the remainder of the application remains unchanged when the application changes its data source.

7.1.2 How

When a request for data is made, the Table Class connects to the appropriate database and gets the data from the database in a record set.

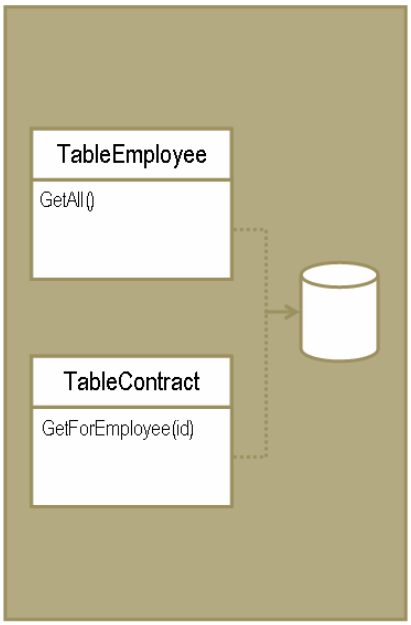

Fig. 7.1 Table Class

The Table Class updates altered data from business entities and business factories. The Table Class doesn't have knowledge of specific business classes.

Table Classes are implemented by creating a Layer SuperType (§ 3.2). The resulting Table Classes for every business entity are static classes. The Table Classes make use of the Query Pattern which will be explained in the next paragraph.

. Property names in the Business class can differ from column names in the database. Therefore every Table class has a describer which handles the mapping between the database and the Table class

7.1.3 Benefits

Table classes give the application database independence. Table classes can individually connect to different kinds of data sources without having any affect on the rest of the application. Because Table classes are implemented using a layer supertype they reduce redundant code in the application.

7.1.4 Consequences

Like said earlier the Table Class connects to the database. Because connecting to the database is a repeatable activity it is best to build a service to carry out this task. In the case of multiple databases the Abstract Factory pattern can be used.

7.2 Query Pattern

SQL can be a complicated language, and many developers aren't particularly familiar with it. Furthermore, you need to know what the database schema looks like to form queries. The Query pattern enables query’s to be written without knowledge of the type of database.

7.2.1 Intent

The Query pattern simplifies and localizes creation of database queries.

7.2.2 How

A Query Object is an interpreter, that is, a structure of objects that can form itself into a SQL query. You can create this query by referring to classes and fields instead of tables and columns. In this way the queries can be written independently of the database schema and changes to the schema can be localized in a single place.

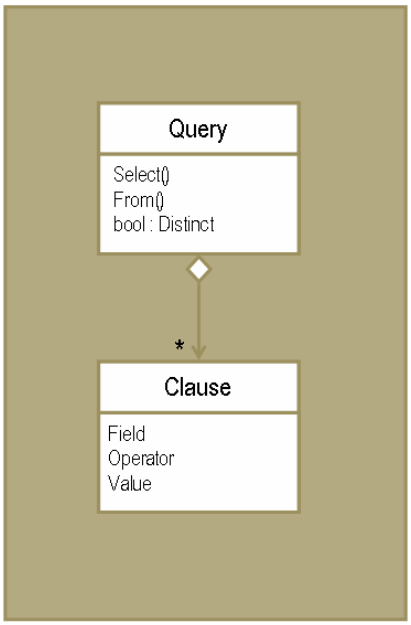

Fig. 7.2 Query Pattern

7.2.3 Benefits

Using the Query pattern abstracts the SQL from the code, making it easier to swap databases.

7.3 Data Factories

A valuable feature of the Trinidad Reference architecture is its database independency. A Trinidad based application can easily switch from database without having to make changes to the code or the queries. The Data Factory facilitates this.

7.3.1 Intent

The Data factory handles al the connections to the database, so the application remains database independent. It encapsulates access to any database from the rest of your application.

7.3.2 How

We have seen earlier that the business class only connects to the table class when in need of data from the data source. The Data Factory gets called from the Table classes in the Data Layer as shown in Fig. 7.3. The Data Factory then connects to the database and retrieves or saves the data.

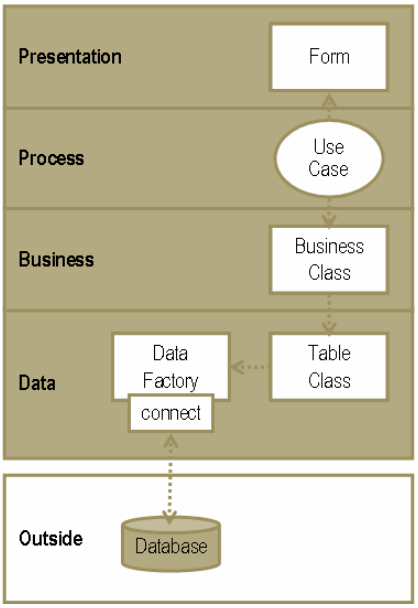

Fig. 7.3 Data Factory

The Data layer contains an interface called IDatabaseprovider. This interface is implemented by an implementation which is plugged into the application according to the plug-in architecture theory. The implementation defines and handles the connection to a specific type of database. A SQLServerprovider class is an example of an implementation connection to a SQL server. The data factories handle the connection to the database but don't execute database queries.

7.3.3 Benefits

The most important benefit of using a Data factory is the database independency it provides.

7.3.4 Consequences

In order to create a structured application the Data Factory should never be directly addressed from layers above the Data Layer. The business logic is isolated in the Business layer and communicates only with the Table class.

7.4 Service Gateway

As we have seen earlier it is possible to create applications with the Trinidad Platform that function as a service consumer. The data source for these applications is a webservice. The Service Gateway pattern is not much different from a Data Factory, except that it facilitates the connection to a webservice instead of a database.

7.4.1 Intent

The service Gateway pattern handles the connections between the service and the application.The service gateway encapsulates access to services and busses from the rest of your application so that the application remains datasource independent.

7.4.2 How

The Service Gateway pattern actually works in the same manner as the Data Factory. The Service Gateway gets called from the Table classes in the Data Layer as shown in Fig. 7.4. The Service Gateway then connects to the webservice and retrieves or saves the data.

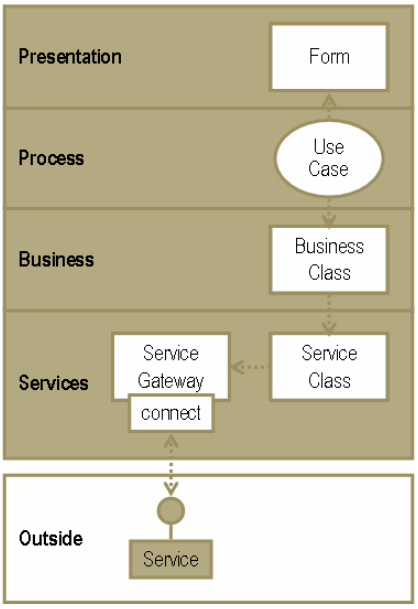

Fig. 7.4 Service Gateway

A Service Gateway can additionally be implemented using an Internal State (§ 6.2). This Internal State object can be used to transfer objects and XML.

7.4.3 Benefits

The Gateway Pattern provides the ability to create service consuming applications (which are making a great march) with the Trinidad Platform. With the aid of the Service Gateway Pattern the Trinidad Platform becomes a flexible development environment.

7.4.4 Consequences

The Service Gateway should also never be directly addressed from layers above the Data Layer. The business logic is isolated in the Business layer and communicates only with the Table class.

8 Patterns

This chapter contains a description of the remaining design patterns that are used in the Trinidad Platform, but have not yet been discussed.

About Capgemini and the Collaborative Business Experience

Capgemini, one of the world's foremost providers of Consulting, Technology and Outsourcing services, has a unique way of working with its clients, called the Collaborative Business Experience.

Backed by over three decades of industry and service experience, the Collaborative Business Experience is designed to help our clients achieve better, faster, more sustainable results through seamless access to our network of world-leading technology partners and collaboration-focused methods and tools. Through commitment to mutual success and the achievement of tangible value, we help businesses implement growth strategies, leverage technology and thrive through the power of collaboration. Capgemini employs approximately 60,000 people worldwide and reported 2004 global revenues of 6.3 billion euros.

The Capgemini Group is headquartered in Paris.

www.capgemini.com

Trinidad Open Source Strategy

Advisory report

Version control

Version	Date	Short description changes
0.1	June 19, 2006	Start Report
1.0	August 15, 2006	Definitive document

Name author(s): Jermaine Jong

Trinidad Open Source Strategy

Name author(s): Jermaine Jong

Company name: Capgemini N.V.
Place: Utrecht
Date: August 15, 2006

Preface / Introduction

This document contains a strategy for an open source Trinidad Platform. The advice in this document is based on research performed during the author's internship on the Trinidad Platform.

Table of Contents

1 The Open Source Model

The open source principle is becoming more and more popular to the developers of software. The Trinidad Platform will be developed and released to customers according to an open source model. This means that customers will use the platform for their own software development projects and at the same time contribute to further development and improvement of the platform. In order to successfully manage this process a clear open source strategy including version- and configuration management is needed. All parties need to be able to easily present and integrate their contributions to the architecture and the framework.

There are benefits but also disadvantages attached to releasing software under an open source model. In order to set up a grounded advice for the Trinidad open source model research about the characteristics of open source is necessary.

1.1 What is open source

Open source software automatically makes people think of “free” software. Actually open source software is much more than that. In [1] the author gives the following definition of open source:

“Open source software is any computer software distributed under a license which allows users to change or share the software’s source code”

The open source Model allows users to develop and contribute to a project voluntarily, which eliminates the need of an actual supplier. The software is shared and can be used freely by anyone. Suppliers can choose to go open source with their products because of the rapid development of their projects that results from the many developers. Communities of developers and contributors are formed where users share knowledge and report bugs and feature requests. Open source doesn’t just mean access to the source code. The distribution terms of an open source program must comply with a number of criteria. These criteria are recorded in the open source Definition. Bruce Perens gives in [2] a short summary of these criteria including a link to the full version, which can be found at http://www.opensource.org/docs/definition_plain.html.

There are a lot of different applications of open source methods, combining properties of markets and communities. In order to set up an open source model it is necessary to know exactly what the open source model is. Most open source models share the following characteristics [1]. This list of characteristics serves merely as an aid to helping the reader form the picture of an open source model.

- *Transparency*, potential contributors need to understand what it is they’re contributing to.
- *Examination of participants only after they’ve got involved*, reducing the barriers to involvement by allowing absolutely anyone to get involved.
- *Low cost and ease of engagement*, many people can get involved at no additional outlay beyond what they already spend on their computing.
- *A legal structure and enforcement mechanism*, open source does not mean a free for all. Instead it depends on a clearly defined legal framework which shapes the incentives for participation.
- *Leadership*, most open source projects have some centralized element of leadership or control that sets the general direction and ethos assigns tasks and acts as an editor, approving changes to the source code.

- *Common standards*, Successful open source projects rely on open, free-to-use standards, and they create new, open, free-to-use standards for their users.
- *Peer review and feedback loops*, the open source collaborative approach manages to produce high quality work because of the many reviews people among themselves.
- *A shared conception of goals*, open source projects may deal with internal dissent about particular choices and directions but there is enough of a common conception of the good to make each project thrive.
- *Incrementalist*, small players can still make useful contributions.
- *Powerful non-monetary incentives*, open source methods contain the ability to replace traditional cash incentives with non-monetary ones like motives of social or personal fulfillment.

1.2 Capgemini's vision on Open Source

The Trinidad Platform is a new development platform within Capgemini. In the short time that the platform exists it has seen a growth of acceptance and interest from other parts of the organization. There are a few projects already carried out on the Trinidad Platform.

Open source projects are booming, even though open source seems like a counterintuitive strategy. Some of the open source supporters even claim that open source projects can withstand Brooke's Law [3] which sounds as follows "Adding manpower to a late software project makes it even later". In an open source project people continuously join or quit development. Even though this claim is not completely true, because most open source projects don't have a schedule so the words "late" and "later" used in Brooks Law are of no meaning here, open source projects are characterized by rapid development by a great number of developers.

Custom software development projects are diminishing. Companies tend to choose offshoring or standard packages, which don't fully fit their needs over custom software development. The main reason for this choice is a financial consideration. In order to compete, custom software development needs to offer benefits over these new development ways. Capgemini believes in the power of the Trinidad Platform and the many benefits it contains for custom software development. Trinidad stands for cheaper, faster, tailor fit custom software development.

The Trinidad platform can be interesting for a lot of custom software development projects. The aim of Capgemini is to spread the Trinidad Platform throughout the world. Capgemini wishes to offer the Trinidad platform to customers in order to let them carry out their own software development projects. If we look at the above, this would fit perfectly within an open source model.

As the founding company behind the Trinidad Platform the Capgemini's name will be spread along with the platform. For support participating companies will turn to Capgemini which has good economical side effects. Capgemini will train and school developers on the Trinidad Platform and deliver support. In exchange, the users of the platform need to contribute any modifications or additions they've made to the platform.

Implementing an open source model has benefits, but doesn't come without risks either. The risks and the benefits Capgemini expects of an open source model are described below.

1.2.1 Technical benefits for Capgemini

There are a number of expected economical benefits for Capgemini from an open source Trinidad Platform. These benefits depend on the chosen business model and open source license. Besides economical benefits Capgemini expects and wishes to achieve the following technical benefits with the release of an open source Trinidad platform:

- Rapid development of extensions to the Havana Framework. The core of the Havana Framework is mature and stable enough, so at the right moment in time to be released as an open source project.
- Quick bug report and fixes, patches, ports to new platforms, etc.
- Growth and evolution of the Trinidad platform and the community.
- With the Trinidad platform Capgemini wants to lean more and more to standardization, because standardization is the key to rapid custom software development. The platform is relatively young and still has a lot of work. However Capgemini knows from experience that most additions, like a specific logging mechanism, are only build when there's a customer who actually has the need for it. Releasing the platform as open source will speed up the development of such extensions rapidly.

1.2.2 Technical risks for Capgemini

The risks for a project to go open source are just as important as the benefits. Before starting an open source project many companies have the same worries about potential risks that such a project may carry [4]. Capgemini oversees the following main risks when starting an open source Trinidad platform:

- Unstructured development process, a lot of open-source software development is still done ad hoc, without good development practice. Most developers start coding before writing down a design [5]. This slows down open source development, while programming is likely to be more regulated and professionalized.

- Fragmentation of the products into incompatible versions, if everybody can issue releases and bring out new versions there can be a spread of different incompatible versions. There need to be a controlling entity in order to structure this.

- Exposing strategies to competitors, the open source model implies sharing strategies and methodologies. This can be a risk, but with good initial support from Capgemini, competitors will rather join development and require support than start an own platform.

- Becoming a product vendor when that is not Capgemini's primary business.

2 Trinidad open source strategy

This chapter will contain a grounded advice based upon the research and study described in the master thesis. This advice will form the strategy for an open source Trinidad Platform.

2.1 License and Business Model

Users or developers of the Trinidad platform might fix bugs or develop additional features. It is important for Capgemini that these modifications are contributed back into the framework so everyone can benefit from a better working framework. Capgemini can enforce this by using the right open source license.

As the study resulted there are a lot of open source licenses available. The Spring Framework uses the Apache license, which allows the users to keep modifications private. Developers of the Spring Framework can make modifications and for example easily sell them. This is not what Capgemini wants for the Trinidad Platform. In order to suit the needs of Capgemini best, the most appropriate license to be chosen is the LGPL license. This license is also used by Hibernate and enforces users to distribute modifications under the same license.

Another characteristic of the LGPL license is that software licensed under it can be mixed with non-free (commercial) software. Because of the open and modular structure of the Trinidad reference architecture it is not unthinkable that the need for connecting with commercial software might occur. The LGPL license is flexible enough to allow this. The full description of the LGPL license can be found at http://opensource.org/licenses/.

Business Model
In order to create revenue from an open source model it is wise to decide on a business strategy. Capgemini wants to offer customers the Trinidad Platform for free but still generate revenue. The approach that Capgemini wants to take in this is to generate revenue by offering training and support.

The companies behind Hibernate and the Spring Framework have business goals similar to Capgemini's. These business goals are accomplished by using a business model called "Support Sellers". These companies have employed a core development team, devoted to working on the project. They provide professional support and training to users and developers of the product. This business model has the advantage that companies using the open source software rather turn to the original developers for support. The more successful the open source project becomes, the more income the business model generates.

Capgemini needs to carry out the Support Sellers business model in order to meet their business goals. Because of the nature of the Trinidad Platform it is likely that the platform will be mostly used by companies rather than individual developers. Especially for companies it is important that the open source project they use is backed up by a company. This gives them a sense of security. When in need of training and support Capgemini will be the obvious company to provide this.

2.1.1 Core development team

In order to support the Trinidad Platform Capgemini will need to form a core development team. Among other things this team will be responsible for releasing new versions of the Framework and coordinating the development of the Framework. The other functions this development team will fulfill will become clear in the following chapters.

2.2 Developers

As we have seen developers are the most important factor in any open source project. Without developers willing to participate and contribute time and effort in an open source project it will surely fail. The study results in a few external factors for developers to contribute to an open source project. Some of these external factors are of a more objective personal nature and can not be influenced by a specific project (like peer recognition). Therefore the most a company can do to make developers comfortable is to provide or stimulate these external factors. In the context of Capgemini and the Trinidad Platform the following issues need to be emphasized in order to make the project interesting for developers and / or customers.

Ready for the Future
Development of software needs to be done faster and faster, because of the growing demands of the customers. In order to reach faster development the development process needs to be standardized. Development within the Trinidad Platform is highly standardized and contains elements that speed up development. Working on a new way of development can boost the joy of programming for developers, and make it interesting for companies to learn the framework and later on even deliver support.

Expanding the developer's skill base
Development within the Trinidad Platform contains a lot of modern development techniques like Use Case modeling, Use Case estimation, Model Driven Architecture and an agile development process. Developers can gather a lot of knowledge on different techniques which make it interesting to participate.

Developer's recognition
Developers like to show their skills, and even get recognition. A reward system within the community could be a good way to support this. Hibernate has a system where users get 'credits' for answering questions from other users. A similar system, with rewards for bug-fixes and patches can attract developers.

Personal needs
Every software development company needs a structured development process. Especially for companies it is interesting to adopt and use the Trinidad Platform.

2.3 Safeguard the quality

The strength of code quality within open source projects lies in its massive code-level peer review. However open source projects lack a structured development process which hurts the quality of code. The benefit of the Trinidad Platform is that it contains a structured development process itself. Development of the framework itself needs to be done according to the same agile development process. Sticking to this development process and keeping in mind the key points below can safeguard the quality of code within the Trinidad platform.

- Develop clear coding guidelines and request from the programmers to keep to this guideline.

- The core development team could asses the code returned by programmers according to a guideline. This implies that the co-coordinator has the right to reject non-conformant contributions, even if they correct a bug or provide new functionality.

- Code re-engineering decisions can be taken by the core development team whenever the project seems to experience problems.

2.4 Community

A successful open source project needs a community where developers can get in touch with each other and provide support amongst them. Users need to be able to ask questions and be kept up to date about news and info regarding the project.

Trinidad User Support Forum
Looking at the Hibernate and the Spring Framework we can learn that it is advisable to distinguish between user support and framework developer's support. The user support forum will allow users to posts and answer questions about Trinidad based software development. A reward system similar to the one Hibernate uses is a valuable extra function to the user forum. It is also preferred that the forum supports multiple languages in order to support users from all over the world. However the initial forum needs to be English, and can later on be extended with additional forums in other languages.

To keep postings in the forums structured it is wise to create a forum etiquette containing guidelines on how to post a question, the amount of additional information that needs to be posted along with the question, etc.

Developer's Mailing list
Many open source projects (Hibernate and the Spring Framework) use a mailing list for developers support. Mailing lists have the benefit that developers stay updated about everything that happens, because every mail that is submitted is received by every member.

It is wise to set up a mailing list within the Trinidad Platform. The Havana framework developers can discuss technical issues in the Trinidad Mailing list. The mailing list is intended for use only by developers actually working on development of the Havana Framework.

Bug tracking
Using a bug tracker is mandatory for every open source project. There are many open source and commercial bug trackers available. The bug tracker is the place where bugs, patches to bugs and feature requests can be submitted. Also for reporting bugs there need to be clear guidelines. Because the Trinidad platform is a new open source project there aren't as many developers reporting bugs yet. Therefore the guidelines don't need to be as strict as for example the ones Hibernate handles. As the project and the amount of developers grow the guidelines can be adjusted if necessary in order to reduce 'traffic' in the bug tracker.

Portal
The Trinidad open source project needs a project portal where users can find information about the Trinidad platform. Documentation like coding guidelines,

manuals and news items need to be available on this site. Also all the other aspects of the community (forum, bug tracker) need to be present on or through this site. As the company supporting the Trinidad development Capgemini needs to host and maintain this site. Possible courses and training regarding Trinidad can also be offered through this portal.

2.5 Version control

The source code of the Havana Framework needs to be made publicly available in order to become a full open source project. However, releasing the source code doesn't mean just offering any body access to modify the code. The source code needs to be placed in a repository (CVS, Subversion, etc) so multiple developers can work together. A good strategy used by Hibernate and the Spring Framework is to make a distinction in rights between developers and users. Developers are initially the core development team from Capgemini containing rights to check out and commit modifications to the source code. In the light of safety and safeguarding the quality of code everyone else has user access. This means that the code can be checked out but modifications can't be committed. Bug-fixes or patches need to be submitted through the bug tracker.

As the project starts to grow the core development team will eventually become too small to keep up with the growth. In this situation the core development team may select loyal users who have committed much to the project through the bugtracker to join the development. These developers will then be part of the core development team and will be given developer rights on the source code.

Backwards compatibility
Open source software doesn't require a company to guarantee backwards compatibility. However it is wise for developers not to vary too much between releases because this will result in dissatisfaction from the users. It is wise to strive to backward compatibility, but guaranteeing 100% compatibility between every release will almost be impossible in the long run. The Spring Framework has all compatible releases up until the present, but hasn't guaranteed this for the future.

It is good for Capgemini to take this as a starting point for the Trinidad Platform. It is wise to strive to maximal backwards compatibility. However if this gets to complicated, mostly when the project starts to grow and expand the strategy of Hibernate can be used. Compatibility and migration problems can be dealt with by means of a migration guide. In this guide all the differences between releases and steps to take when upgrading need to be explained. This doesn't give the users 100% backwards compatibility but compensate for this by guiding the users through the migration process.

2.6 Change management

The proposed change management process for the Trinidad Platform is already partially enforced by the version control system. Only members of the core development team can directly modify the code. All the changes, bug-fixes or feature requests from other developers are submitted through the bug tracking system. The core development team will decide on the correctness of the bug-fix, the quality of the code and reject or approve the change accordingly.

Both Hibernate and the Spring Framework use the same change management process. This is a safe way to secure the integrity and quality of the code.

2.7 Release management

In order to frequently issue new releases of the framework a release management process needs to be configured. Hibernate has a checklist that developers need to follow before issuing a release. The Spring Framework doesn't seem to have a fully structured process. However in both projects the core development team is responsible for the release. This team decides on for example the bugfixes that will be implemented in the next release.

Pre-release testing
Before publishing a new release it should be tested according to a few criteria like code tests, acceptance test, etc. The core development team needs to make sure that every new feature or bugfix is tested. The advice here is to set up guidelines for testing that can be directed back to the developers. Every developer needs to bundle a patch or bugfix with a test.

This doesn't mean that the core development team doesn't need to test anymore. It is the responsibility of the team to gather and structure all the tests, carry out regression tests and perform the final check. Acceptance tests can also be part of the pre-release testing process.

Release approval
A member of the development team needs to coordinate the pre-release activities en give the final approval for issuing a release. This member can function as a controlling mechanism on top of the pre-release testing process. It is not clear how this process is done in Hibernate or the Spring Framework but for the Trinidad Platform it would certainly be a valuable addition.

Distribution
After a release is formed it needs to be distributed. The latest version of the code can of course be obtained from the source code repository, but it also needs to be made available in a more user friendly manner. The first step that needs to be taken is to notify users and developers of the new release. The best place to announce the release is in the community; within the forum, the developer mailing list, the frontpage of the website, etc. Like Hibernate and the Spring Framework the best place to distribute the framework is the Trinidad Platform website.

2.8 Success factors

In order to set up a successful open source Trinidad Platform the open source success factors that have been studied need to be taken into consideration. T Some of these open source success factors are closely associated to the open source characteristics that are discussed in this chapter. However the advice to Capgemini is to use the full list of open source factors as a supplement to the open source characteristics.

Community

C1.Marketing / Promotion of the open source project
C2.Central community with support fora and communication
C3.Respecting and accepting ideas and viewpoints of others, if appropriate
C4.Stability, openness, transparency and fast response times for these communication and information exchange activities
C5.Conferences, developer meetings and workshops about a special subject.

C6.A sense of community together with clear dispute resolution mechanisms – in a democratic sense of decision making.
C7.Initiation of self-organization processes like support of new participants by experienced community members.
C8. A central vision, most projects are better off with one or two open minded people guiding the project and maintaining a single coherent vision.

Development

D1.Short intervals for new releases and application and testing by as many users as possible
D2.A clear declaration and identification of beta-and stable releases.
D3.A comprehensive documentation of the code and a roadmap for development
D4.General preparations (by a core team) for and following discussion of requirements and targets of further development in the community.
D5.Definition of preferences and priorities of certain projects.
D6.Avoidance of monolithic code.
D7.Permanent quality management.
D8.A comfortable opportunity for distributed software-development with a concurrent versions system.
D9.Permanent bug-fixing
D10. Avoid over-design*;* Aiming for too much abstraction and flexibility at an early stage is a waste of time.

Economical

E1.Professional Open Source, generating revenue by choosing the right business model.
E2.Full-time developers backed by a commercial company.

Bibliography

[1] Mulgan, G., Steinberg, T., Salem, O. *Wide Open, Open source methods and their future potential*, Demos, 2005.

[2] Perens, B. *The Open Source Definition*, http://perens.com/Articles/OSD.html. Visited at: 15-08-2006.

[3] Brooks, F. *The Mythical Man- Month*, Addison-Wesley, 1975.

[4] Hecker, F. *Setting Up Shop: The Business of Open-Source Software*, IEEE Software, 1999.

[5] Wilson, G. *Is The Open-Source Community Setting a Bad Example?*, IEEE Software, 1999.

About Capgemini and the Collaborative Business Experience

Capgemini, one of the world's foremost providers of Consulting, Technology and Outsourcing services, has a unique way of working with its clients, called the Collaborative Business Experience.

Backed by over three decades of industry and service experience, the Collaborative Business Experience is designed to help our clients achieve better, faster, more sustainable results through seamless access to our network of world-leading technology partners and collaboration-focused methods and tools. Through commitment to mutual success and the achievement of tangible value, we help businesses implement growth strategies, leverage technology and thrive through the power of collaboration. Capgemini employs approximately 61,000 people worldwide and reported 2005 global revenues of 6,954 million euros.

The Capgemini Group is headquartered in Paris.

www.capgemini.com