Architecture of SureThing Information System

Bogdan Dumitriu
Marjolijn Elsinga
Sébastien Raveau
Reinier Vis

January 25, 2005

Contents

Chapter 1

Background information

1.1 Scope of this document

In June 2004 two famous Dutch insurance companies (Zeker & Vast and ForSure) merged into SureThing. As a consequence of this merging, the two companies needed to integrate both their hardware and software into a single, unified, system which could serve for running the business of the new SureThing. This document comes to describe the architecture which we propose as a support for this hardware/software integration.

The Zeker & Vast Company had quite a different company strategy than the ForSure Company. Zeker & Vast has cooperated with some hundreds of intermediaries for offering proposals and taking care of claim handling for various amounts of customers. The intermediaries communicated with the company through its Call-Center. The target of this company was the middle class society. The ForSure Company did not have a Call-Center, but had five account managers providing a single point of contact to each of their wealthy customers, with each account manager managing about 100 customers. The company's target was the upper class society.

While Zeker & Vast already comes with quite an advanced IT system, ForSure has a rather rudimentary one, based mostly on the use of various Microsoft Office tools by each account manager separately, with no standards and no integration. Both of these systems are described to a certain extent in the Request for Architecture which we have been presented with and which can be made available separately. Some of the current situation (especially the one at Zeker & Vast) will also be discussed in this document as a starting point for our changes.

After the merging, the SureThing Company operates at five different locations in The Netherlands: three former locations of Zeker & Vast, one location of ForSure and the new headquarters. The management desires that the new system provides a fully shared IT application landscape that can be used by every employee regardless of the location at which the employee resides. Most of the hardware will be running at the old Zeker & Vast headquarters in Amsterdam, which is connected to each of the other locations by 2 Mbps lines.

The management of the newly created SureThing Company has a few goals which guided the developing the architecture presented in this document:

- they would like to ensure an increase in profit and future growth of the market share of SureThing, as well as make the customer handling process more efficient and less costly, by serving Zeker & Vast's customers directly, besides dealing with the intermediaries (e.g. by providing web access to insurance proposal requests).
- they would like to provide real-time proposal calculation, as opposed to the current situation where requests for proposal calculations are honoured only one day after they are made.
- they would like to have an information system which allows them more flexibility in defining the company's products.
- they would like the new system to use a so-called Generic Output Component which Zeker & Vast currently has in development for all paper communication.
- they would also desire a preferred supplier strategy for hardware (and software).
- since they are concerned about the social and cultural consequences that the changes in the IT situation will trigger, they decided to adopt an incremental strategy for developing and moving onto the new system.

1.2 Assumptions

The architecture presented in this document is based on a number of assumptions which we have made (note: in real life these would have been clear facts which we would've found out during the process of requirements elicitation). They are stated below.

1. Currently, Zeker & Vast only works with intermediaries. We make this assumption in order to be able to limit the number of stakeholders which we consider.
2. We assume that management's "incremental strategy" translates to a desire for an architecture which involves several transformation steps to reach the final goal. This is a vital assumption for our architecture, as based on it we introduced several phases for changing the current system, striving to ensure that in each of these phases the system is fully operational and behaving correctly.
3. We assume that CHIPS & BuRP do not use the Oracle database directly in order to get / store their information, but rather use UCIS as an intermediary. We assume this since one of the points from the Desired Situation section in our assignment description read: "CHIPS and BuRP will need to be upgraded to be able to interface with the new UCIS". Consequently, we believe that if CHIPS/BuRP had been using the database directly, they would no longer have had to interface with UCIS (either the old or the new one).

4. On the other hand, we assume that STIFF accesses the Oracle database directly (i.e., not through the UCIS system). Since the phrase "Information from UCIS's Oracle database is read directly [by STIFF]" can be interpreted in either way, we simply chose to give this interpretation.

5. Also regarding STIFF, we assume that it is started indirectly by UCIS, by placing certain information in the Oracle database. This assumption was made because the way in which communication takes place between UCIS, STIFF and the Oracle database was not clearly mentioned in the description. In connection with this assumption, we also assume there is some kind of cron daemon (see [cron]) which starts STIFF every night (since we are being told that "STIFF is a batch-oriented system that runs nightly").

6. We assume that the information about clients and proposals of the ForSure Company is still available even if the account managers have left the company. We make this assumption since it is unlikely that company policy would've allowed the account managers to take this information with them, supposedly leaving the company without some of the records regarding its business.

7. As a worst case scenario, we assume that STIFF is written in Cobol, running on quite old hardware, which would make it difficult to change in order to accept direct requests for proposal calculation, leaving us with the only option of rewriting it completely. We need STIFF to accept direct requests because that's the only way we can make it compute proposals in real time (see also next assumptions).

8. We assume that STIFF in its current (Cobol) form can be run on new hardware as well. As nothing was mentioned about the internals of the STIFF component in the assignment description, we had to assume this in order to be able to provide solutions for making STIFF run in real time.

9. We assume that the old STIFF software is not multithreaded. This assumption is used in the physical view in order to justify the type of hardware that needs to be bought.

10. We assume that computation of a single proposal by STIFF can be done in terms of a few seconds on new hardware (and thus allow for a real time response to the customer). We need this assumption because otherwise we cannot ensure real-time proposal calculation.

11. We assume that the SureThing company wants different access rights for different groups of employees.

12. We assume that the current hardware configuration is as follows: most of the IT system (including UCIS, STIFF and the Oracle database) will continue to run in the old headquarters from Amsterdam (where it was running before the merging) and that 2 Mbps lines connect this IT center to the other four locations of the SureThing Company. We had to assume this as there is no information about how the system will be run after the merging and this was the most reasonable assumption, since it involved the smallest amount of reorganization.

13. We assume SureThing does not own a spare rack mount cabinet at the moment. Clearly, no mention of this appears in the description, and we need the assumption for cost computation.

1.3 Estimations

Number of requests for proposal

One of the main aspects the system has to be designed for, is being able to sustain a certain number of requests for proposals.

Measurements over the past year, as mentioned in the Request for Architecture document, have indicated that about 140,000 proposals were issued as a result of a request made to Zeker & Vast. The document also indicates that this number represents about 30% of the total number of calls made by customers to the Call-Center. This leads to the conclusion that about 470,000 customers have contacted the company during a year. Since we can expect quite an increase in the number of requests for proposals once the company provides the possibility of requests being issued also directly, through the Internet, we estimate that probably the percentage of issued proposals will increase from 30% to about 60%. We assume this because it is a known fact that customers, once no longer being faced with a human being at the other end of the line, will tend to request proposal calculations even without any intent whatsoever of purchasing an insurance. The number of requests is thus expected to increase to 280,000. If we add to this the number of customers the company will gain simply by offering another means of contact - the Internet - in addition to the Call-Center, we expect somewhere around 350,000 customers which will request proposals to be computed. Since the number of customers ForSure currently has is about 500 (5 account managers x about 100 customers each), we can safely go with the 350,000 figure as the final estimation (since the 500 extra will make no difference).

This number of customers implies an average customer load of 1,000 customers a day, which (as we assume they are more or less all in the same time zone) will all be accessing the system probably during daytime, i.e., 10 to 12 hours. This yields an average load of about 1 or 2 customers / second, and a maximum load of probably somewhere around 10 to 20 customers / second.

Number of developers needed

We do have quite a large system to deal with, on one hand, but most of the components are already available, on the other, which means that the development work involved will not be extremely high. We have therefore created this architecture on the estimation of having about 8 to 10 developers available for the duration of the implementation. We believe that this number is sufficient to ensure the development, deployment and initial testing (with possible corrections) of the system in the 11 months which are available until the end of 2005.

Chapter 2

Stakeholders

In this part of the architecture document the different stakeholders will be described and their concerns listed. There will be a distinction between minor concerns and major concerns, denoted with - and + respectively. A minor concern is one that is important to some extent, but can be (partially or totally, depending on the concern) dropped. A major concern, by contrast, is one that is of great importance to its stakeholder and which (s)he would like to see in the final system. However, compromises can be (and have been) made even in terms of major concerns. A compromise does not imply a total disregard to a certain concern, but rather providing a solution which might only partially address it. The following stakeholders have been considered:

- Users: staff of the SureThing Company
- Acquirers: SureThing Company
- Developers and Project Manager
- Maintainers
- System Administrator
- Management (of the company doing the development)

2.1 Users

Description

The users are the people that will be working at the SureThing Company. This group will be interested mostly in things like how friendly the new system will be, if they have to learn new things and how big the changes will be that are being made to the user interface. The people visiting the web site of the SureThing Company are not viewed as part of this stakeholder because it would be difficult to sit with them around the table to discuss their concerns. We assume that their major concern would be whether the web site gives them the information they are looking for in an usable manner or not. To cope with this, we have transferred this concern to the acquirers, since they are stakeholder for which the customers' concerns are most important.

Concerns

The concerns of the users are:

- - Availability during work hours: the system needs to be fully operational especially in the period of time when they need it, namely during work hours. Little or no downtime of the system is thus desired in this period.
- + User friendliness: the users want to work with a easy to use, intuitive and clear system.
- + Response time: the users are interested that the system takes as little as possible for performing its actions and responds quickly.
- - Understandability, learnability and operability: this concern expresses the wish of the users for the new system to be as easy to understand, use and switch to as possible. The users hope not to have to learn too many new things in order to be able to work with the system.

2.2 Acquirers

Description

The acquirers are the managers of the SureThing Company. This group is mostly interested in making sure that all the goals put forward in the assignment description are achieved. The system which the present document describes needs to be beneficial to their business and in many ways better than what they have now. Although they have a wide range of concerns, they are primarily interested in making sure that the resulting system will address their (company's) needs in a proper manner.

Concerns

The concerns of the acquirers are the following:

- - Realization of the system is in conformance with their requirements.
- + Scalability: as the number of customers increases, the number of requests for proposal calculation will increase as well and so will the use of most of the other components of the system (such as the amount of output of the GOC, for example). Therefore it should be possible to scale the system as the need arises in order to cope with the increased load. Scalability is viewed mostly in two aspects: web site availability and responsiveness and how the company's hardware deals with the load.
- - Suitability: the system should address the company's needs and not the other way around.
- + Security: the new system needs to provide at least as much security as the current one does, but preferably even more. The stakeholder underlines that the insurance business deals with highly sensitive data which should stay in the company (as opposed to out in the open).

- Cost: the system needs to be improved at a reasonable cost in terms of money and human resources, it needs to be finished by the end of 2005 (but preferably sooner) and it should be as adaptable as possible in order to prevent high costs in the future.

+ Progressive evolution of the system: the old system should be turned into the new one in several phases (also see our assumption on this).

+ Correctness and usability during transition: during the transition of the old system to the new one, the system has to be usable and behave as expected the whole time.

+ Interoperability: the system has to be based on and work as much as possible with the already existing components, in order to keep costs low and also in order to avoid the need for extensive retraining of people.

- Compliance to standards: the software of the system needs to be able to connect to other software if necessary and, most importantly, the web site needs to be browser independent (in other words, it needs to run on any of the major browsers in use today).

+ Fault tolerance: the system needs to be available 24 hours a day, 7 days a week.

- Clarity of web interface to customers: this is in fact a concern of the users of the web site, but since we have not identified them as stakeholder, we make this a concern of the acquirers. Thus, the acquirers want their product to be as appealing to and usable by their customers as possible.

2.3 Developers and Project Manager

Description

The developers and the project manager are the people that will coordinate and implement the whole transition of the old system into the new system. They are especially interested in understanding how the current system works, what kind of changes they have to do to it, how these changes are scheduled in time, how the new system is to be structured, what kind of constraints they have to take into account and many more.

Concerns

The concerns of the developers and the project manager are as follows:

+ Modularity: the system needs to be as well structured as possible, so that all components of the system can be developed, tested and replaced independently of each other.

- Security: the developers are interested to know what kind of security the system has to provide and what this security implies on their part. They want to know which components of the system are most sensitive and what should be done in order to protect them.

- Scalability: the stakeholder is concerned with what parts of the system have to be scalable and what has to be taken care of in order to ensure this scalability.

+ Technical constraints: what are the technologies that have to be used in order to develop/update the system? What knowledge does the use of this technologies imply on their part? Also, it is of interest to know what underlying systems are currently running the in the IT environment of the company, especially if these are to be maintained in the new system.

+ Understandability of the design: they would prefer the design to be as intuitive as possible so as to be able to focus more on development and less on deciphering the architectural description.

- Testability: the system has to be tested, so the stakeholder is interested to know which parts have the highest correctness requirements so that they can test them extensively.

- Development time: developers are worried about how strict the final and intermediate deadlines are. They would like to be told the consequences of exceeding development time.

2.4 Maintainers

Description

The maintainers are people that will have to solve any bugs or changing requests of the system once it is released. They, as persons, might overlap to some extent with the developers, depending on company policy, but their concerns are nevertheless distinct. This group is interested if the system is adjustable to the future needs of the company, if it is easy to change, if it is easy to track down errors that might appear in it and if it is appropriately documented.

Concerns

The following concerns are typical for the maintainers:

+ Changeability: if changes need to be made, this must be done to a limited number of components and there must be a documentation to know how. This goes partially hand in hand with the modularity concern of the developers as a modular system is also easier to change.

+ Testability: the system needs to be easily testable in order to find bugs and ensure its correctness now and in the future.

2.5 System Administrator

Description

The system administrator is the person (or group of people) that is initially responsible for deploying the software on the existing hardware configuration according to specifications and then for making sure that the system is operational at all times. Also, the system administrator is concerned with making backup copies of important information so that it can be restored in case of hardware malfunction. Finally, the system administrator also has to understand what hardware and software changes have to be done once the current configuration is no longer sufficient to deal with the load.

Concerns

The concerns of the system administrator will be mostly related to:

+ Security: what has to be done in terms of configuration so that the system is secure (firewalls, protocols, configuration files, etc.).

+ Recoverability: it should be easy to spot what the problem with the system is and it should be easy to solve the problem.

+ Fault tolerance: is the system supposed to provided 24/7 access? If so, what has to be done from his/her part in order to ensure this?

- Installability: the system and the new components should be easy to install.

- Manageability: analysis of the running system should be possible and modifications to it should be straightforward.

2.6 Management

Description

Management represents the interests of the developing company and has to make sure that the proposed system can be done within budget, has to make sure that the customer (i.e., the acquirer stakeholder) is pleased with the result and also has to make sure that costs are kept as low as possible and, more importantly, not exceeded.

Concerns

The following are the concerns belonging to the management:

+ Feasibility of constructing the system: management needs to be convinced that the system, as described in the architecture, can be put into practice within the money and time budget.

- Changeability: it is important to management that the system be easily changeable as they believe the SureThing company might have changing requirements in the future and that their company will again have to work on the software, so it would be best if that implied easy to do changes.

+ Profit: they would like the architecture to be created in such a way that implementation thereof can be done with as low costs as possible in terms of time, personnel, resources, training and money so that their profit can be higher.

- Costs might be exceeded: management wants to diminish the risk that the costs allocated to the project are exceeded, as this would conflict with their concern for making profit.

Chapter 3

Requirements

As a result of our discussions with the stakeholders we have decided upon the requirements described in this chapter as the defining ones for our architecture.

3.1 Functional requirements

The functional requirements display the needs and expectations of the stakeholders in terms of software and hardware demands. We would like to point out that most of the functional requirements of the system are already met by the existing software and therefore we only list here those requirements which are new. It should be self-evident that the new system will include all the functionality of the old system (possibly providing a changed implementation for it). Here are then the extra requirements:

- Web access to customers: One of the major functional requirements is the requirement of web access for customers. The situation now only allows customers to call the Call-Center and get their information there. The new system needs to supply a web access for customers also. At this website there will be information about the company but also the possibility to make a request for a proposal calculation.

- Discount for customers requesting proposals via the web site: The company wants to have a kind of discount for the customers that make a request for a proposal calculation via the web site, to stimulate this kind of proposal requests instead of proposal requests through the Call-Center.

- Stability for web request for proposal: If a customer visits the web site and makes a request for a proposal calculation through the web site, it should be possible to freeze the prices and discounts of that moment until the time that the customer actually signs a contract. It is not appreciated when the customer makes a request for a proposal calculation at one time, and thinks about it for a couple of days, then returns to the web site and finds out that in the mean time the same request will give a much higher outcome. Therefore a period of price and discount freezing is suggested (e.g. a week).

- Use the GOC component: The GOC component is not yet in use, because some miscalculations by one of the preferred suppliers of the IT development organization. The GOC is predicted to be available in March 2005. By then, all the paper communication must start using the GOC component.

- Access control to system functionality: There need to be different access levels for the different groups of people accessing the database and the other components of the system. The system needs to distinguish especially the web site users from the companies users, because the information in the database is confidential.

3.2 Quality attributes

The quality attributes of the system are a reflection of the stakeholders concerns. Because there are so many concerns involved in the system and some of them are contradictions, a selection needed to be made. For each group only the major concerns (denoted with +) where selected and of this pool of concerns we have chosen the six most important quality requirements:

1. Interoperability: we view this as the most important quality attribute, because the system needs above all to be able to interact with the components the company already has and build upon these.

2. Fault tolerance (24/7 availability): One of the vital concerns of the acquirer is that the system must be available for the company's customers 24 hours a day, 7 days a week. They see this as strategic for their expected growth. The system to be developed, then, needs to be robust enough in order to ensure that, even in the case of isolated hardware failure, the functionality will still be there for the customers. Only in the case of extended failure (e.g. a fire in the building, with most servers burning down) is it acceptable to have an nonoperational system.

3. Scalability: we expect the company to grow fast, as a result of the merging and of the company's future Internet availability, so the new system needs to be scalable to be able to deal with a bigger amount of customers in the future. We expect a steady increase in the number of customers and failure to cope with this number of customers would be highly counterproductive for the company. Therefore we have set this as an important quality aspect of our architecture.

4. Security: because of the confidential kind of information the company deals with every day, in combination with an access to this information database via the web site, the system needs to be designed with security in mind. We see this as an important attribute which needs to be taken into account because its lack might have very negative effects on the acquirer and its customers.

5. Changeability: this attribute (together with its related one, modularity) is of interest to a great part of our stakeholders, therefore we have decided to also take it into account as prioritary. Another reason for selecting it

is the desire to take an incremental approach in the transformation of the system, which would obviously require **changes at each step**. It is only natural, then, to see this as a quality attribute to have in mind in order to ensure a smooth process.

6. Response time: The last significant attribute on our list is response time due to a firm demand on the part of the acquirer of having a system capable of computing proposals in real time. During discussions it has become clear that this is a very requested attribute especially in what web interaction is concerned so we were convinced to include it.

Of course quality attributes can be associated with the other concerns as well, but we believe that if we are able to make a system that satisfies these six quality requirements, then every stakeholder will be satisfied and will be able to use the system in a right and fulfilling way.

3.3 Constraints

The constraints that guide our design process are as follows:

- Cost: It appears from the discussions that both acquirers and management wish to keep costs quite low for this project, so we will define it as a constraint for us and, as a consequence, strive to come up with solutions which are as less costly as possible and, most importantly, make sure that the development of the system does not exceed these costs. This embraces the cost in terms of money, time and people needed to accomplish the system.

- Correctness and usability during transition: The whole transition of the old system to the new system takes place in several phases. During each of this phases the components that are already finished should be usable and should be working correctly. We list this as a constraint for us because each phase in particular has to be able to be materialized into a runnable entity. In fact, this constraint is largely related to the fact that SureThing wishes to have an incremental upgrading of their system.

- Oracle & J2EE: In terms of technologies that have to be used, we are constrained to using the Oracle database platform as it is already purchased by the company and it is neither likely nor necessary that it will be changed. On the other hand, we will impose the use of the J2EE platform as a constraint ourselves, largely due to the fact that it is already currently in use by the UCIS system, but also due to reasons which we explain later in this document.

Chapter 4

Guide to the Architecture

4.1 Incremental approach

We have created our architecture in such a way that the transformation of the old system into the new one takes place in several phases. This is necessary because of the desire to have an incremental upgrade and is also profitable for the acquirer as it ensures early benefit of the changes that are already made. In addition, it is better for testing and user acceptance of the new system in small steps. As mentioned, during this transition the system will have to be working correctly and be usable the whole time. The transition, as explained, will consist of several phases, each of them including a number of steps. There will also be steps that span across 2 phases, but these represent isolated cases, as most steps will be completed in one phase only. Below, we describe the steps we have divided the upgrade into.

1. **UCIS refactoring to benefit from on-demand proposal request processing by STIFF**: We do this step as soon as possible because it implies quite small changes and because it is necessary in order to allow the customer (which, at first will be just the Call-Center employee, but once step 5 is also completed, will also be the regular customer) to benefit from real time proposal calculation

2. **Create STIFF wrapper to emulate future on-demand protocol on old STIFF system**: STIFF currently calculates the proposal request over night. After the addition STIFF will be able to do calculations right away. We achieve this by adding a wrapper which will start STIFF on request. This might not be real time yet, but still it will be considerably faster than before

3. **Migration of ForSure data from Office documents to Oracle database**: Currently, the data of ForSure is spread in a non-standardized way among a number of Microsoft Office documents. Clearly, it is impossible for an automated system to work with such data, which implies that all of it has to be transferred into the Oracle database of Zeker & Vast, where it will be stored in a structured manner

4. **Rest of UCIS refactoring with GOC integration**: We refactor the business logic of the UCIS system so that all the business logic currently

existing on the client-side is moved to the server-side, leaving nothing but the presentation to be done on the client. This step is necessary before we can advance to the next one, in which we transform the presentation layer into HTML. During the refactoring, we also need to have in mind that all output has to be now forwarded to the GOC component, which is also introduced as part of this step

5. **Porting of UCIS client GUI to website**: This will be the step at the end of which it will be possible for customers to access SureThing's system directly, through the Internet

6. **Update CHIPS and BuRP to interface with the new UCIS**: As UCIS is modified, CHIPS & BuRP will have to be updated in order to use the new interfaces of UCIS (if necessary)

7. **Rewriting of STIFF in Java for scalability and caching of data**: While STIFF is still in its current form, running multiple instances of STIFF at the same time is not possible (reasons are explained in the process view), which means that it cannot be scaled to run on multiple machines at the same time. In order to allow such scalability, we need to rewrite STIFF from scratch and, since we are doing it anyway, we propose to rewrite it in Java since this way it would integrate easier with UCIS (which is also written in Java).

   We will also want to make the response time even smaller by caching product information from the database in STIFF's memory. There is a point to this since we expect product information to change very rarely (in terms of days). There is thus no point in rereading such information from the database every time a proposal has to be computed. However, in order for this system to work, STIFF has to be notified when the product information in the database changes. Creating such a notification mechanism is also the purpose of this step.

8. **Migration of product data from ProD to Oracle database**: Alongside with rewriting STIFF, we naturally want to drop the text database (ProD) and move all the data found therein into the Oracle database, together with the customer information. Also connected to this step is the necessity of creating a database schema for the products that is flexible enough to allow for future product customization

9. **Use Oracle stored procedures to centralize the SQL code**: This step is concerned with creating stored procedures in the Oracle database which provide all the data access that is now randomly spread through the sub component of the system. In this way, we would ensure that the application will not be sensible to future changes in the data model, as long as the stored procedures are updated accordingly

Steps 1 through 3 are planned to be finished by the end of phase 1, steps 4 through 6 by the end of phase 2 and the other steps by the end of phase 3. We also include an additional phase (phase 4) which is meant for testing the system and deciding if the load is properly dealt with or hardware additions are necessary. This division of steps into phases will also be discussed in more detail in the development view.

4.2 Guide for stakeholders

The interesting part of this document for the users is the scenarios view (ref).

The part of this document that is of interest of the acquirers is the Physical view (ref).

The parts of interest for the developers and the project manager of this document are the Logical view (ref), the Process view (ref) and the Development view (ref).

The parts of interest for the maintainers of this document are the Logical view (ref), the Process view (ref) and the Development view (ref).

The parts of this document that are of interest of the system administrator are the Process view (ref) and the Physical view (ref).

The parts of interest for the management are ...

Chapter 5

Viewpoints

5.1 Logical Viewpoint

Because of the fact that all the stakeholders have different requirements and different concerns, not all those concerns can be covered in one view. The next view is the Logical view, which is part of the "4+1 View Model of Software Architecture". This chapter will first describe which stakeholders will be addressed by this view. At second, the concerns that are covered with this view are mentioned, and after that we will show diagrams with explanation on how the system should be implemented to cover all these concerns.

Stakeholders

The stakeholders that are addressed by the logical view are:

- Developers and the Project Manager
- Maintainers
- Management and the Acquirers (partially)

Concerns

The concerns that are covered by the logical view are:

- Will the system have a progressive evolution?
- How is the correctness and usability guaranteed during the transitions?
- Will the system be interoperable?
- Will the system be modular?
- Will the system be testable?
- Will the system be changeable?

Modelling techniques

The logical viewpoint can be modelled by using any of the following UML diagrams:

- Class- / Object diagrams
- Sequence/collaboration diagrams
- State-chart diagrams
- Activity diagrams

Library reference

For more details about the logical viewpoint, you are referred to [Kruchten1995].

5.2 Process Viewpoint

The process viewpoint is meant to give an image of the running system, as described by the architecture. It comes to complete the logical viewpoint by modelling the processes which, through their interaction, define the behaviour of the system. The viewpoint explains issues such as synchronization, concurrency, communication protocols, threads of control and so on.

Stakeholders

The stakeholders the process viewpoint is relevant to are:

- Developers and project manager
- Maintainers
- System administrator (to some extent)
- Management (to a very small extent)

Concerns

The concerns addressed by this viewpoint are:

- How does the system behave dynamically?
- What are the processes that co-exist in the system and how do they interact?
- What are the protocols used for communication?
- How is the system scalable (in terms of process multiplication)?
- What is influenced by changes in the behaviour of a process?
- Which processes have to be able to communicate directly with others?

Modelling techniques

The process viewpoint can be modelled by using any of the following UML diagrams:

- Sequence/collaboration diagrams (usu. including active objects)
- Component diagrams
- State diagrams
- Activity diagrams

Naturally, any other kind of diagram the architect might find appropriate is welcome if accompanied by appropriate explanations.

Library reference

For a slightly more extended description of the what the process viewpoint should contain, as well as for an example of a process view, you are referred to [Kruchten1995].

The description of UML diagrams can be found in a variety of sources (both printed and on-line), one of which is [Fowler2000].

5.3 Development viewpoint

Stakeholders

The stakeholders that are addressed by the development view are:

- Developers and the Project Manager
- Maintainers

Concerns

The concerns that are covered by the development view are:

- Is the final system modular?
- Is the final system testable?
- Is the final system easily changeable?

Modelling techniques

The development viewpoint can be modelled by using any of the following diagrams:

- Package/component diagrams
- Gantt diagrams

5.4 Scenarios Viewpoint

The scenarios viewpoint is redundant with most of the other viewpoints, its purpose being that of providing a guide to reading the other architectural views. By describing a number of relevant, representative use cases of the system and explaining how their goal can be achieved by using the proposed architecture, the scenarios viewpoint can also be regarded as a proof that the architecture is a good solution for the creation of the required system.

Stakeholders

The stakeholders the scenarios viewpoint is relevant to are:

- Acquirers
- Management
- Developers and project manager
- Maintainers (to some extent)
- System administrator (to some extent)

Concerns

The concerns addressed by this viewpoint are:

- To what extent does this architecture comply with the requirements?
- How can the proposed architecture be used for achieving user goals?
- How (in what order) should the other views be read through?
- Where (in what view) can relevant information about a certain aspect of the system be found?
- What actions of the system are implied by (some relevant) user goals?

Modelling techniques

The scenarios viewpoint is usually described using UML use case diagrams, accompanied by a narrative description of each use case. The sentences in the description will be in the active voice, present tense, describing an actor successfully achieving a goal.

Library reference

For a understanding the scenarios viewpoint in a larger context, you are referred to [Kruchten1995].

5.5 Security Viewpoint

The purpose of the security viewpoint is to provide developers and system deployers with vital information about how the system has to be built and later on deployed, such that all the security concerns expressed by the customer are handled. Security information can vary from instructions about what protocols to be used to network configuration and yet to suggestions about authentication and authorization mechanisms. The viewpoint should express enough details about what has to be done, but should also be clear enough so that the acquirer stakeholder can understand it and be convinced that the system security will indeed correspond to his/her level of expectation.

Stakeholders

The stakeholders the security viewpoint is relevant to are:

- System administrator
- Developers and project manager
- Acquirer
- Maintainers (to some extent)

Concerns

The concerns addressed by this viewpoint are:

- What protocols have to be used in order to secure communication?
- What parts of the system have the highest security requests?
- How should the network be configured in order to protect the system?
- How is authentication/authorization to be done?
- How secure is the system?
- What parts of the system are less secure and why?

Modelling techniques

The security viewpoint will mostly be narrative, but appropriate diagrams can be used where they could make understanding easier. Examples of such diagrams:

- (annotated) Deployment diagrams
- (any kind of) Network diagrams
- UML State diagrams (for explaining security protocols)
- Database entity-relation diagrams (for, e.g., explaining access control mechanisms)
- ... and many more

Chapter 6

Logical View

The logical view represents the classes and the objects of the system. Since we don't want to dig too specific into the class structure of the different components, we will show some diagrams of a higher level of the system. The diagrams show how the different subsystems are related to each other, how they are connected with each other and also how they evolve as the system evolves into it's final form.

6.1 Information

In this view, we want to show the way in which the system evolves from the current situation into the desired situation where both companies are merged, we first have to give a graphical representation of the current system. From here on, we improve the system in certain steps to make the system more modular and changeable. To show these changes, we decided to point out three deadlines, at which we expect that the system is adapted according to this document. These deadlines will be addressed to as *Phase 1*, *Phase 2* and *Phase 3*

We also think that at each phase, the system will be still fully functional and working correctly. The whole architecture is set up in a way that all the functionality of the prior phase, will still be available at the end of the phase. If -for instance in the case of STIFF- it wasn't possible to refactor the whole application in the first phase, but we did want to provide on-demand calculations, we designed a wrapper around the system to provide this functionality as soon as possible. The wrapper will be removed when we refactor the STIFF system, but now with little effort, we can keep the functionality of the system as a whole intact.

The figures in this view show the modularity of the system. Every subsystem has its own functionality, which makes it easy to make changes to one of the subsystems separately. Because of this modularity, the testability and the changeability of the system is also good.

6.2 The current system

The current system consists of several smaller programs which interact with each other. The following components are identified in the current architecture:

- UCIS: Universal Customer Information System
- ProD: Product Database
- STIFF: System for Tariff Information For Fee-calculations
- ProFi: Proposal Financials
- Print Proposals
- CHIPS: Claim Handling and Payment System
- BuRP: Bulk Refund Product

For more detailed information of these systems, we would like to point to the specifications given in the Request for Architecture of the SureThing company

Logical View

Below you see the logical representation of the current system.

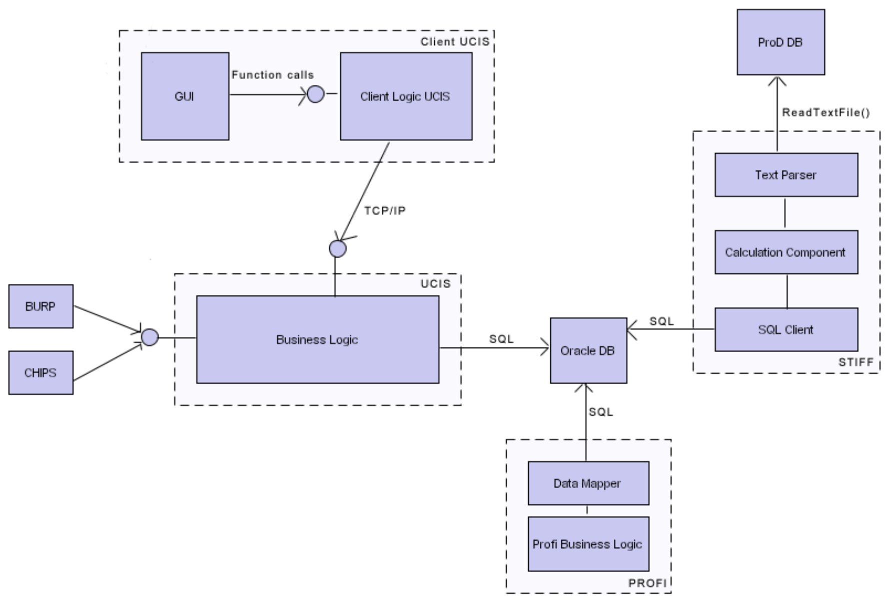


Figure 6.1: Image of the current system

Additional information

As you see in the picture, the current system provides two databases, one for the product information, and one for the customer information. Since the ProD is a plain-text database, it is desirable to move this information into the main (relational) database. The current STIFF system is also fairly outdated. It is written in COBOL, which makes changing this application very hard. The system also cannot provide real-time calculations of proposals to the UCIS system. All the calculations are currently done in batch-mode at midnight.
The client software has gained more and more logic into its components. These calculations should be incorporated in the business logic at the UCIS server component. In the most desirable case, we want the client software only to be able to display the Graphical User Interface, and the client software should not have any logic implemented at all.

The matter in which the changes should be scheduled and applied to the system are addressed in the following sections.

6.3 The system at Phase 1

In the first iteration we want the STIFF system to behave as a real-time component. Because we will not make any big changes to the STIFF logic yet, we decided to make an extra wrapper around the STIFF system, which starts the calculations on request. The calculations (and reading of the text-based ProD) are still the same, the only improvement here is that the calculations are no longer done in batch mode, but real-time and on demand.

Logical View

Below you see the logical representation of the system at Phase 1. See figure 6.2

Additional information

We choose to use RMI as the interface to the STIFF component, because it makes the implementation more easy. With Java RMI you don't have to worry about creating sockets for the connections, because this is already implemented in the RMI specification. The serializing of the objects and the object relations is also easier to implement using RMI.

6.4 The system at Phase 2

In the second phase of the system we will mainly refactor the business logic of UCIS. When we refactor the components, we also have to make CHIPS and BuRP make use of the new UCIS components that are available.
The other main change is that we transform the Graphical User Interface (GUI) of the old UCIS system into a JSP website.

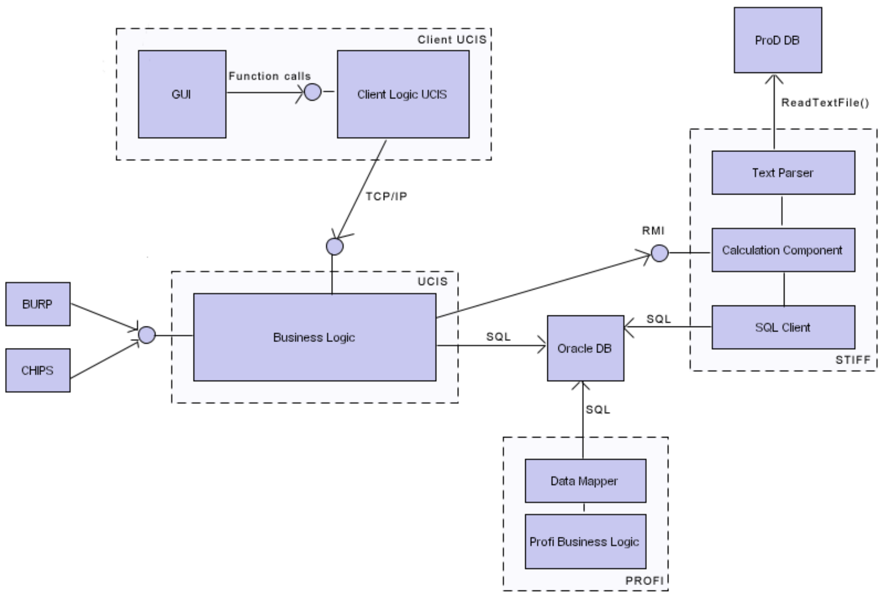


Figure 6.2: Logical View after phase 1

Logical View

Below you see the logical representation of the system at Phase 2. See figure 6.3

Additional information

The main transformation in this phase involves UCIS. This system is refactored to conform to the layered architecture. All the business logic will be moved to the Business Layer and the the Data Mapper will be implemented in the Data Layer. As presentation layer we will use a JSP website, which is served to customers through the web server and the Internet. Because of this, anyone with a web-browser installed on their computer will be able to calculate their own proposal. (Access to detailed customer information is restricted of course).

The BuRP and CHIPS components do also have to be updated, in a way that they will from now on use the interface that UCIS provides to get the desired functionality for these systems.

6.5 The system at Phase 3

In the last phase we will refactor the whole STIFF component in Java. A notification system will be introduced in STIFF, and UCIS has to make use of this notifier at the end of this phase. The Product Database will also be migrated into the Oracle Database from here on.
In this last step we also include the Generic Output Controller (GOC).

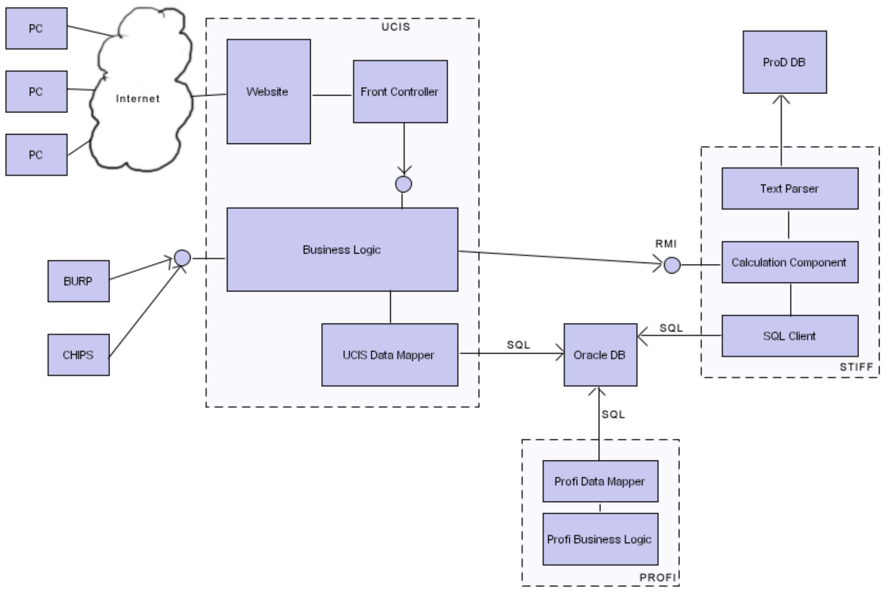


Figure 6.3: Logical View after phase 2

Logical View

Below you see the logical representation of the system at Phase 3. See figure 6.4

Additional information

The whole STIFF component will be refactored in Java. The old system of STIFF was completely written in COBOL, which made it very hard to make changes in. To provide real-time proposal calculations, the system had to be scalable to some extent. We can only make STIFF scalable, if we refactor it completely to Java.
The calculations that have to be calculated in real-time, take a lot of resources of the STIFF system. To ensure real-time calculations of the proposals, we have to make the STIFF component scalable. That means, that is the load on the server becomes to high, we can add another server to STIFF, so that from that point on the load is distributed among the two servers. If the amount of requests still grows and more servers are needed, new servers can be easily added to the system.

Because of the fact that the product information doesn't change very often, it is not desirable that each STIFF server requests the latest product information from the database at each calculation request. To decrease network traffic, and to decrease the load on the database server, it is best to store this information in the memory of the server, and only when the product information changes, that the STIFF servers gather the new information from the database.

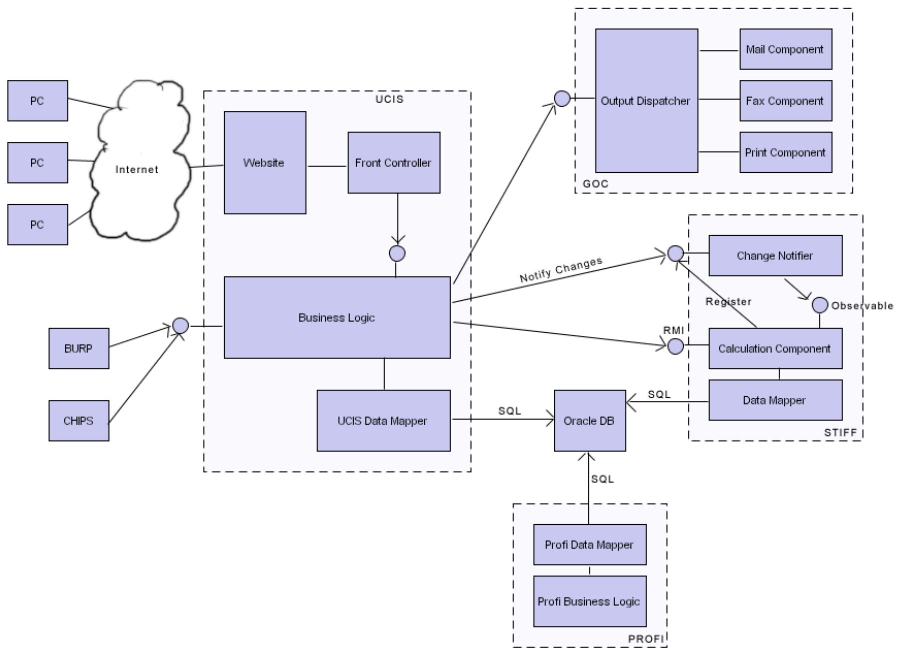


Figure 6.4: Logical View for final system

Chapter 7

Process View

The process view, as the name indicates, corresponds to the previously documented process viewpoint. As such, it is mostly targeted at the system developer and, in part, to the project manager categories of stakeholders. To some extent, it can also be of use to the system administrator stakeholder, for a better understanding of the interactions in the system.

7.1 Overview of processes

The first series of models in the process view (figure 7.1, figure 7.2, figure 7.3, figure 7.4) illustrates the highest level configuration of the processes existing in the system. Each of the figures describe this configuration at a particular phase in the evolution of the system.

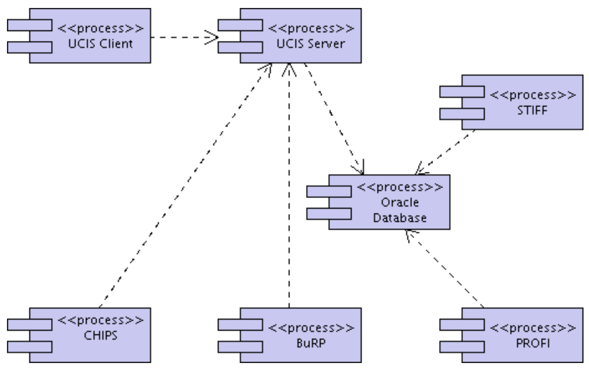


Figure 7.1: Configuration of processes, phase 0

As can be seen, figure 7.1 shows the processes in the system in phase 0 of development (the current state of the system). Note that in this phase UCIS is divided in a client and a server process, with each client process running on a separate machine (currently, machines used by the Call-Center employees) and the server process running on a single machine (currently located in the central headquarters in Amsterdam). Another relevant thing to notice is that the STIFF process provides no means of interaction for the other processes. Use of the STIFF process is done solely by means of the Oracle database, both in order to make proposal computation requests and to collect results.

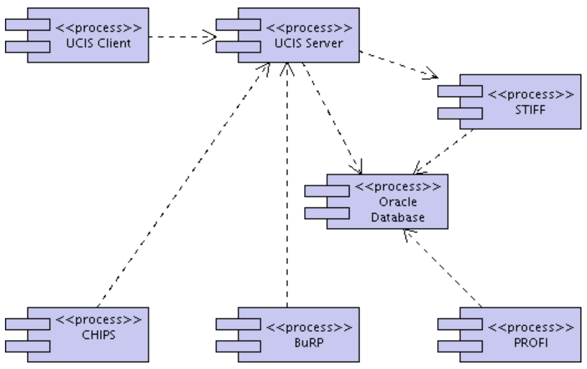


Figure 7.2: Configuration of processes, phase 1

This problem is addressed in two steps: first, in phase 1, a simple wrapper around STIFF is to be provided, which will allow direct interaction of the other systems (particularly UCIS) with it. This modification is envisioned at this earliest phase since it:

- involves a very small amount of work;
- can contribute significantly to an increase in the productivity of the company, since a proposal could thus be computed right away, instead of having to wait 24h for it to become available.

The second step involves a full rewrite of the STIFF system in Java, again making sure that it can accept requests from other systems. Since this implies a lot more work than the first step, it is only planned to be finished by the end of phase 3 (see development view). The difference to be noticed between the phase 0 configuration of processes and the other phases is the additional line connecting UCIS to STIFF, which implies STIFF's availability for direct requests (i.e., no longer through the Oracle database).

Another difference to be noticed between phase 1 configuration and phase 2 configuration is that UCIS is transformed from a traditional, proprietary protocol, client/server system to a web-based client/server system, with the old client/server processes from figure 7.2 being replaced by the new browser/web server processes from figure 7.3. This transformation is justified by two requirements:

- the requirement to provide web-based access for customers;
- the (implicit) requirement to refactor the UCIS system so that the entire business logic runs on the server, leaving only the presentation layer for the client (browser) to handle.

Going from phase 2 to phase 3 implies, in terms of processes, the introduction of the GOC process which is to be used for generating all output. This means

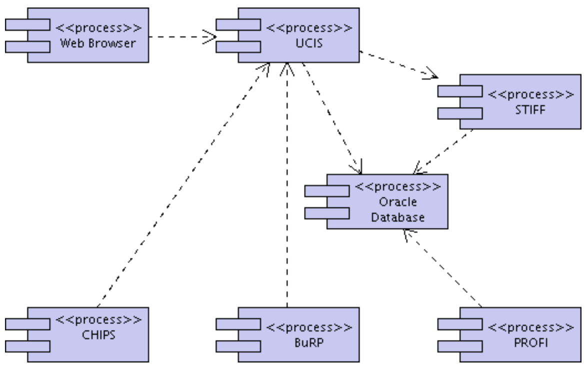


Figure 7.3: Configuration of processes, phase 2

that all the requests for various types of output which are made through the PROFI and the UCIS system will no longer be handled by the respective process, but will instead be forwarded to the GOC process. This modification comes to meet the requirement that all output has to be done through the GOC component by the end of 2005. Also in phase 3 the STIFF process is supposed to be rewritten completely in Java, using a multithreaded design, but this is not visible at this higher level. This distinction is, however, made clear in the next section.

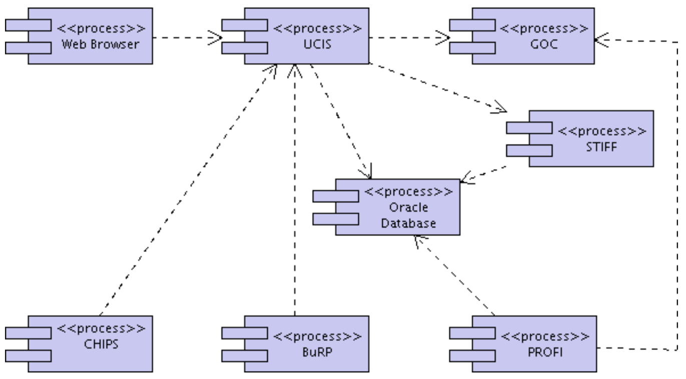


Figure 7.4: Configuration of processes, phase 3

7.2 Process interaction

This section goes into slightly more detail about how the processes described in the previous section actually interact. As our topmost quality requirement is interoperability, we show here how the processes (are supposed to) interact at the end of each of the three phases in the evolutionary development of the system. Throughout, we try to point out by means of the diagrams how interoperability is ensured at each phase of development.

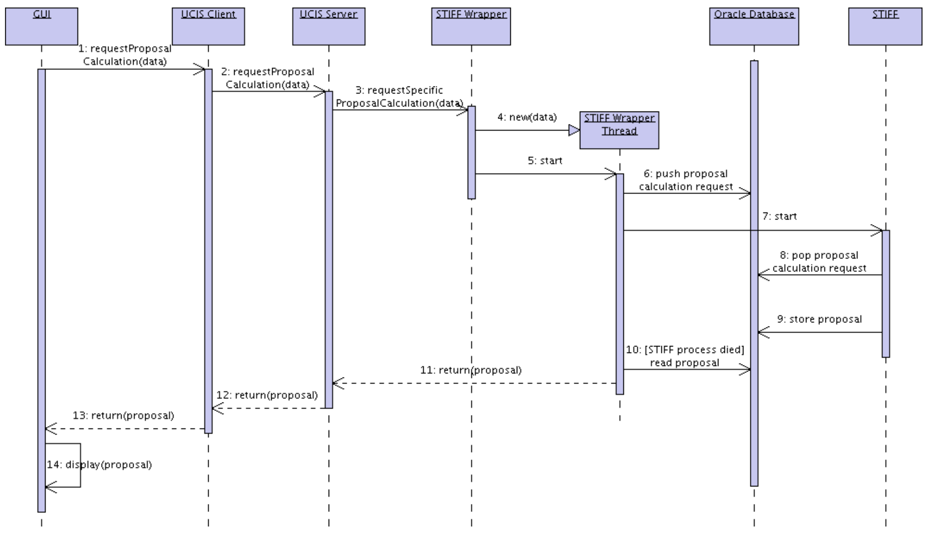

GUI
UCIS Client
UCIS Server
STIFF Wrapper
Oracle Database
STIFF
1: requestProposal Calculation(data)
2: requestProposal Calculation(data)
3: requestSpecific ProposalCalculation(data)
4: new(data)
STIFF Wrapper Thread
5: start
6: push proposal calculation request
7: start
8: pop proposal calculation request
9: store proposal
10: [STIFF process died] read proposal
11: return(proposal)
12: return(proposal)
13: return(proposal)
14: display(proposal)

Figure 7.5: Process interaction, phase 1

Process interaction at the end of phase 1 is illustrated in figure 7.5. Here, the Call-Center employee requests a proposal calculation by means of the GUI. Existing proprietary protocol communication takes place over TCP/IP between the UCIS Client and the UCIS Server. When the request arrives, the UCIS server calls the newly introduced (in this phase) STIFF Wrapper, making a request for a specific proposal calculation (after previous interpretation of the generic request made by the client). The wrapper will start a thread for handling the request. This thread will then conduct the process described below to get the proposal and then return it to the UCIS server. The UCIS server will send the result back to the UCIS client, again by using a proprietary protocol over TCP/IP. Here is, then, the process by which the wrapper thread conducts the calculation:

1. The wrapper thread places the proposal data in the Oracle Database, marking it as uncomputed (e.g. by setting a boolean value).

2. The wrapper thread starts the STIFF system.

3. STIFF will run as before, i.e. it will read all entries in the proposal table and will check if any of them have to be computed (e.g. by looking at the boolean value), make all computations, and place the result(s) back into the database. All wrapper threads will have to coordinate so that only one instance of STIFF is started. Any of the common synchronization mechanisms (locks, semaphores, etc.) can be used for this purpose. The reason why we don't want more than one instance of STIFF to be started is because otherwise we might have two or more instances of STIFF computing the same proposal which, especially since the used algorithms are so complex, would be counter-efficient.

4. The wrapper waits for STIFF to end execution and then reads the results from the database and returns them to the UCIS Server.

In terms of process interaction, the only changes brought by phase 2 are that the UCIS system no longer communicates using a proprietary protocol, but instead by using HTTP. This is a consequence of the UCIS system becoming a web-based system as the main change of phase 2. This can be seen in figure 7.6. Although depicted in the diagram, you are referred to [Fowler2003] for a complete description of the *Front Controller* pattern (which is what is used here in order to handle HTTP requests), where you can find all the necessary information also in what the process itself is concerned. The reason for using the *Front Controller* pattern in this system is because we want to be able to handle any common task (e.g. authentication) in a single place instead of having to duplicate it for each separate page which the UCIS provides.

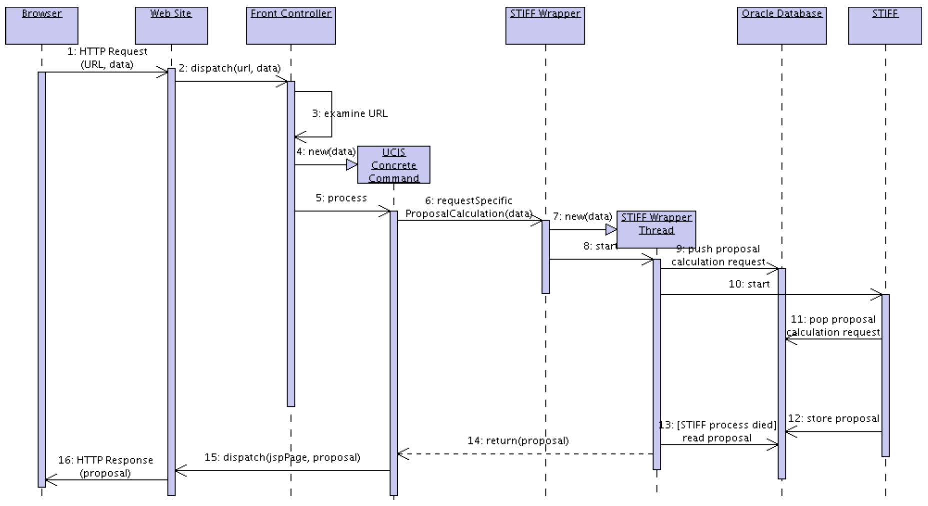

Browser
Web Site
Front Controller
STIFF Wrapper
Oracle Database
STIFF
1: HTTP Request (URL, data)
2: dispatch(url, data)
3: examine URL
4: new(data)
UCIS Concrete Command
5: process
6: requestSpecific ProposalCalculation(data)
7: new(data)
STIFF Wrapper Thread
8: start
9: push proposal calculation request
10: start
11: pop proposal calculation request
12: store proposal
13: [STIFF process died] read proposal
14: return(proposal)
15: dispatch(jspPage, proposal)
16: HTTP Response (proposal)

Figure 7.6: Process interaction, phase 2

As a remark, in what the UCIS system is concerned, it will naturally run as a multithreaded system, but all multithreading/synchronization issues are fully handled by the J2EE application server, so details about such issues are not provided in this document as they are not seen as being part of the application itself. On the other hand, for scalability reasons, UCIS is to be modified (if necessary) in such a way that multiplication of the whole system on any number of machines is possible. This implies that any session information has to be stored in the database so that clustering can be done in a transparent manner. More explicitly, a (possible) load balancer might (and probably will, due to randomness) dispatch requests pertaining to the same logical session to different instances of the UCIS server. By using the *Database Session State* pattern this can be easily handled so that it does not become an issue. For further information on the aforementioned pattern, you are referred to [Fowler2003].

Finally, the process interaction by the end of phase 3 is shown in figure 7.7, where you can see that communication with the STIFF process can now be done in a much easier way, as we no longer need the wrapper. As also mentioned in the logical view, by the end of phase 3, STIFF is expected to have been completely rewritten in Java, using RMI to make itself available for direct requests from the UCIS at any time. The reason for choosing RMI is that by using it, all inter-process communication is simplified to mere remote method invocations, which allows for faster (and also cheaper) development. An additional argument supporting the decision to use RMI is that the amount of communication (measured here as the number of remote calls) that is needed is quite small (one remote call per customer request), so then the performance of the overall system will not suffer due to an otherwise significant delay entailed by remote calls. This loose dynamic coupling (i.e. small number of remote calls) between UCIS and STIFF, combined with our resolve of achieving a high degree of scalability, also explains why we are placing STIFF on different machines than UCIS. As a remark, notice that the process (involving the STIFF Wrapper) described as necessary in phases 1 and 2 can now be discarded altogether, as it is no longer needed.

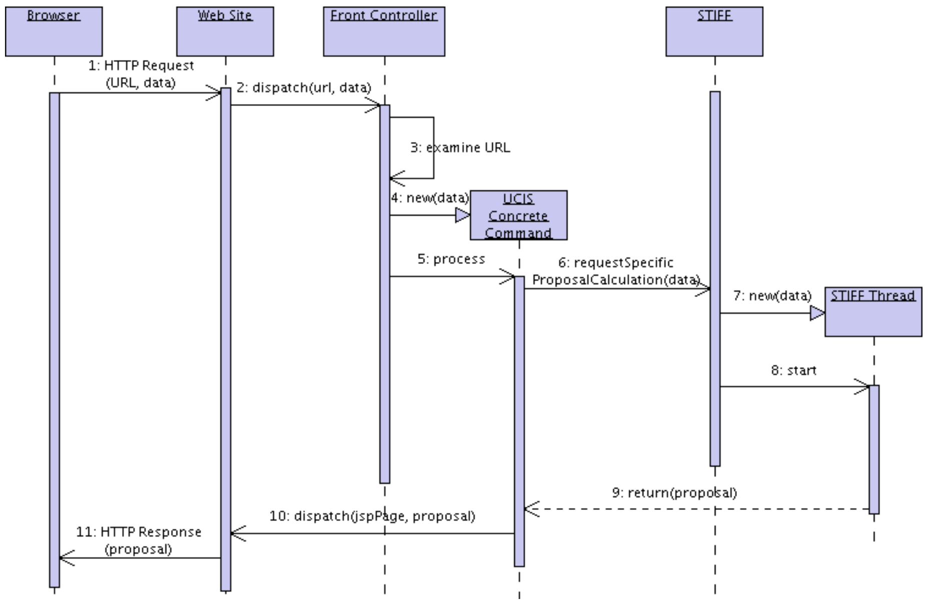


Figure 7.7: Process interaction, phase 3

7.3 The STIFF System

Last but not least, as one of our main goals is scalability and as scalability comes into play especially when discussing the STIFF system (since this is the part of the whole application with the largest processing needs), it is essential that the rewriting of STIFF in Java (planned for the 3^{rd} phase of development) meets the following requirements:

- any number of STIFF processes can be run in parallel on any number of machines (to ensure scalability), without any need for communication between them (to ensure transparency);
- STIFF processing is done in a multi-threaded manner so there is a chance, when STIFF runs on multi-processor machines, that separate threads can

be scheduled by the operating system to run on different processors, thus allowing for scalability of the number of processors.

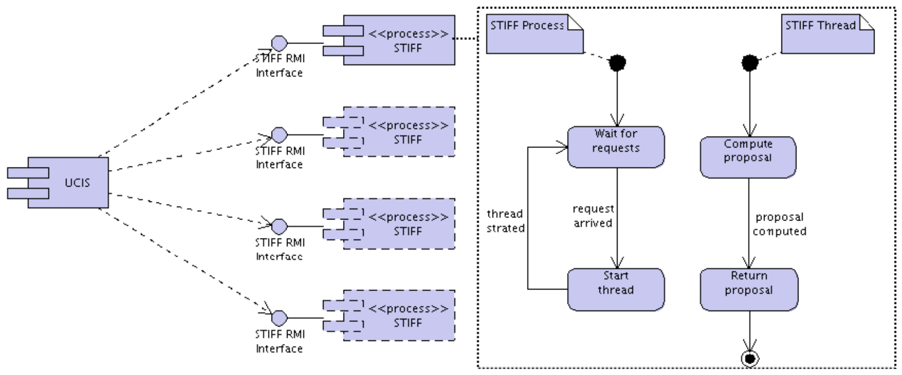


Figure 7.8: Overview of the final STIFF system

The first requirement implies that no synchronization is necessary between STIFF processes. This is feasible as each proposal calculation is essentially independent of the others and as now UCIS makes proposal requests directly to STIFF (i.e., not by setting values in the Oracle database). The latter means that the danger that more STIFF processes/threads might compute the same request for a proposal calculation no longer exists (as we expect UCIS to properly coordinate what request goes to what STIFF system).

The second requirement has already been briefly mentioned in figure 7.7, where we show the main STIFF process spawning a new thread to deal with each request. Achieving this is also necessary in order to meet the response time quality attribute which we have set as a goal. By having multiple threads we expect that the average response time will be increased due to possible parallel processing (on multi-processor machines).

Figure 7.8 summarizes this discussion, while the physical view completes it with scalability aspects which are relevant from the physical viewpoint.

Chapter 8

Development View

The placement of components into a system is not always the same as the diagram of the Logical View. It might be possible that two programs make use of the same set of packages. The Development View shows how these packages/components are distributed among the system. It also shows at which time the different subsystems have to be finished in order to succeed in developing the system as a whole.

8.1 Information

The development view on this system is nearly the same as the logical view, but some of the components that form the final STIFF system have to be developed as a part of UCIS.

The STIFF system will do many proposal calculations during a single day. If the product information inside the database doesn't change very often (which is the case here), it is useless to query the database at every calculation request for the same product data. This is why we decided that the product data should be kept in memory of the STIFF system, and only if the product data is changed, that the STIFF systems refresh their data with the new product information. Because the actual amount of STIFF systems is never known by UCIS, there has to be some functionality which can tell all the STIFF systems to update their memory. This is done by the STIFF Notifier.
The STIFF Notifier, which is part of the STIFF system, has to be developed as a part of the business logic of UCIS. The STIFF Notifier will receive a message from UCIS when the product information is changed in the database. The Notifier now sends a broadcast message to all the STIFF systems to update their memory, and all the systems behind the load-balancing mechanism receive the update message.

In figure 8.1 you can see the deployment of the components of UCIS and STIFF.

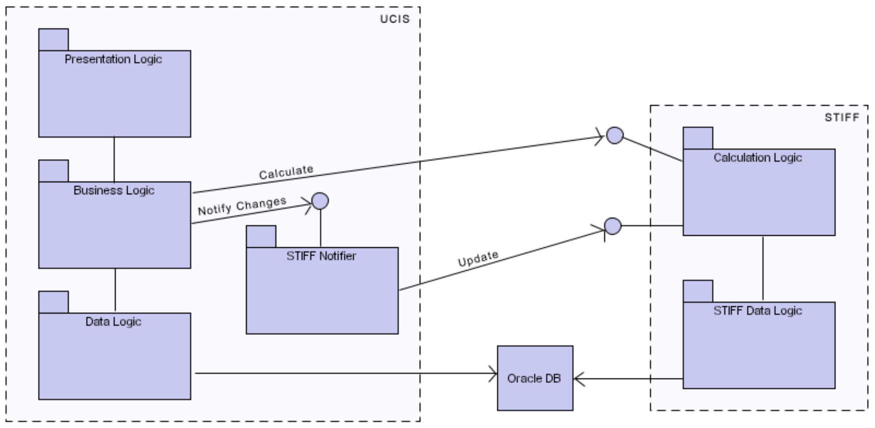


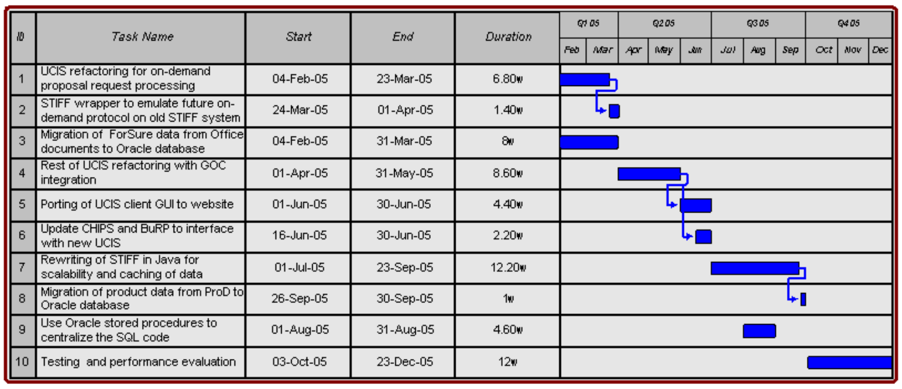
Figure 8.1: Development View of UCIS and STIFF

8.2 Evolution in time

The time schedule in which the system has to be developed, can be shown best in the next Gantt diagram (see figure 8.2). In the first phase, the system has to be altered in a way that it will perform direct requests (on demand) instead of batch-wise calculating the proposals. To provide this functionality, both UCIS and STIFF have to be changed. UCIS will have some minor changes, just the ones for benefiting of the real-time calculations of STIFF. STIFF on the other hand, will need some major refactoring to be done to ensure real-time calculations. The whole system has to be transformed from Cobol into Java. Since this is too big of a step, we will wait with the refactoring of STIFF until the third phase. In this first phase, STIFF will receive a wrapper which starts the system on demand, and that just lets STIFF make use of its old components.
In this first phase, we also want to insert all the customer information of the customers of ForSure into the Database of Zeker en Vast. Since the Oracle database isn't depending on ProD for storing the actual insurances (the ProD doesn't keep track of pricing fluctuations for instance), the policies of the customers of ForSure can be added to the customer database without any problem.

In the second phase, the whole UCIS system will be refactored. It will get a new user interface which will be accessible through the Internet, and all the business logic that had become part of the client program will be put inside the main UCIS system again. Also, when the new GOC component is available, it will be added to the system in this phase. The last action in this phase is to make sure the CHIPS and BuRP system make use of the new interfaces that will be provided in the new UCIS system.

In the third phase the STIFF system will be refactored. From now on, the STIFF system will be totally scalable, and adding extra servers to provide more computational power will be very easy. When STIFF is refactored, it will no longer make use of the old ProD database. All the product-data will be stored in the main Oracle Database from this point on. When changes are made to the products, the STIFF systems will be automatically be notified by the UCIS system of that.



ID	Task Name	Start	End	Duration
1	UCIS refactoring for on-demand proposal request processing	04-Feb-05	23-Mar-05	6.80w
2	STIFF wrapper to emulate future on-demand protocol on old STIFF system	24-Mar-05	01-Apr-05	1.40w
3	Migration of ForSure data from Office documents to Oracle database	04-Feb-05	31-Mar-05	8w
4	Rest of UCIS refactoring with GOC integration	01-Apr-05	31-May-05	8.60w
5	Porting of UCIS client GUI to website	01-Jun-05	30-Jun-05	4.40w
6	Update CHIPS and BuRP to interface with new UCIS	16-Jun-05	30-Jun-05	2.20w
7	Rewriting of STIFF in Java for scalability and caching of data	01-Jul-05	23-Sep-05	12.20w
8	Migration of product data from ProD to Oracle database	26-Sep-05	30-Sep-05	1w
9	Use Oracle stored procedures to centralize the SQL code	01-Aug-05	31-Aug-05	4.60w
10	Testing and performance evaluation	03-Oct-05	23-Dec-05	12w

Figure 8.2: Gantt diagram for the development of the system

Chapter 9

Physical view

9.1 Phase 1: making STIFF real time

Since the current asynchronousness and overall slowness of the STIFF system clearly appears to be an obstacle to maximizing business at SureThing, we recommend taking care of this part of the architecture first. This will improve the company's yield at an early stage, and therefore will probably help financing the rest of the operation, as opposed to taking care of the STIFF system in the last.

The best way of significantly improving the system's performance as soon as possible is to provide some kind of quick fix at first, so that proposal calculation requests are processed *on demand*, until the complete redesign of the STIFF system is ready for deployment.

Another advantage is that by making this in two steps it is possible to make a minimalistic investment in hardware at first, on which to have the transitional STIFF system running, and then evaluate the need for extra computing power required by the real time STIFF system.

Cost: about €3000

The most appropriate for a STIFF cluster would be to buy rack servers, so that space is not wasted by arraying several bulky tower servers.

However, an initial investment of about €1000 will be necessary to buy a rack mount cabinet of capacity between 24 (which is usually the minimum) and 48 space units (U's). Server racks are then designated as 1U, 2U and 4U depending on their size.

A study of all the rack server solutions proposed by DELL has been done, and it would seem that the PowerEdge SC1425 corresponds the best to this project's needs: it is the most affordable DELL performance server and it fits in only one rack space unit (1U). It comes with either one or two processors, but since the old STIFF software was not designed to run multithreaded it will drain all the power from the first processor and none of the second one. Therefore we recommend buying a first PowerEdge SC1425 containing only one processor but the strongest one available (Intel Xeon EM64T 3.6GHz, 1MB Cache, 800MHz FSB), for a price of approximately €2000.

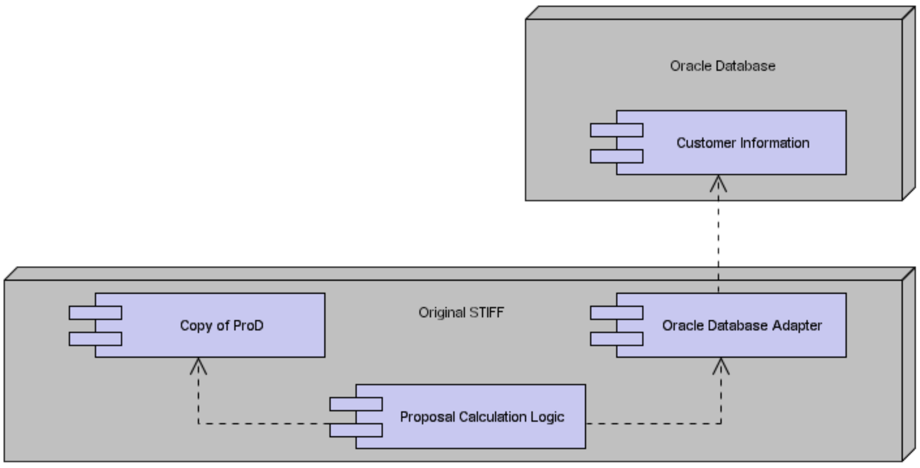


Figure 9.1: Current (standalone) STIFF, running overnight

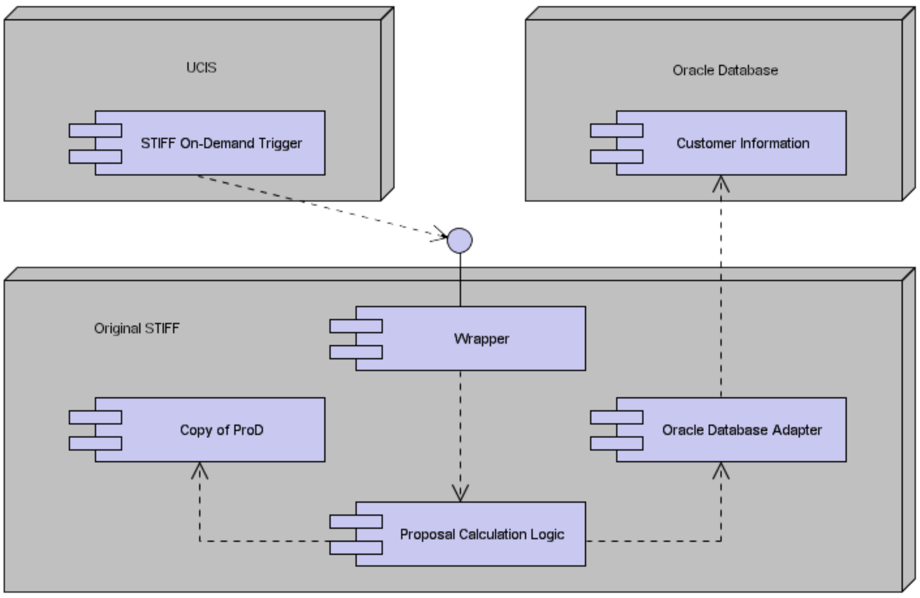


Figure 9.2: On-demand STIFF, triggered by UCIS via a temporary wrapper

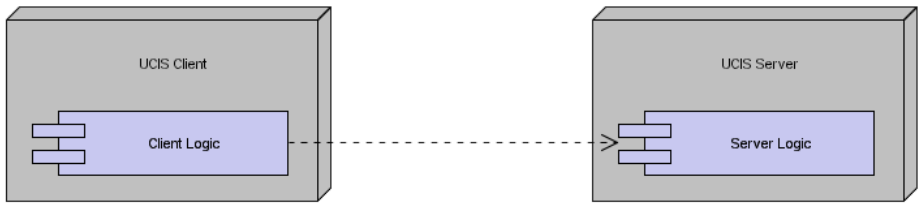


Figure 9.3: The UCIS system, with business logic on the client side

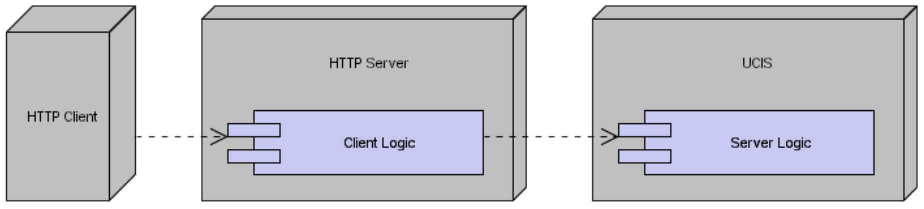


Figure 9.4: The UCIS system, with centralized business logic

Note that once the new STIFF software is ready, it will be able to fully take profit of a multiprocessor system; thus other solutions such as the dual processor version of the PowerEdge SC1425 rack server based on two similar but slower (Intel Xeon EM64T 2.8GHz, 1MB Cache, 800MHz FSB) processors will be more efficient, and at the same time cheaper: around €1700 for this particular solution.

9.2 Phase 2: building the web front-end

Currently, the Call-Center employees use a custom-made UCIS client for Windows that contains a significant amount of business logic. This is difficult to maintain because each time that portion of business logic has to be updated, it has to be updated on every single computer where the UCIS client is installed.

Consequently, we recommend centralizing that logic in a website, which in turn will also free SureThing of all the concerns towards compatibility of the UCIS client with the various operating system, or various operating system versions the UCIS users may be running. It helps SureThing into achieving the Extended Enterprise model by providing a web infrastructure to the company, a requirement for maximizing cooperation with partners.

In order to switch from the UCIS client application for Windows to a web-based user interface, an investment in dedicated web servers is required.

Cost: about €1500

The main concern on the web servers will be reliability, and not so performance. Therefore we recommend buying entry-level rack servers such as the DELL PowerEdge 750, which comes for about €1100 when based on a single processor (Intel Celeron processor, 2.4GHz, 128K Cache, 400MHz FSB) and 1GB of memory.

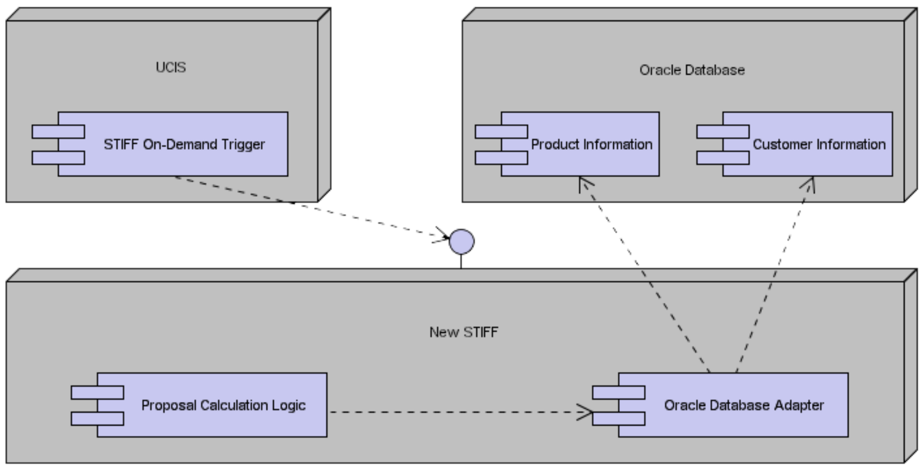


Figure 9.5: The final STIFF system

A single PowerEdge 750 will be sufficient to handle the initial load generated by the traffic of ex-users of the UCIS client for Windows on the website. And once the company's commitment in the e-commerce world broadens, attracting more and more visitors to the website, it will be easy to add one or several other PowerEdge 750 units to the rack mount cabinet shared with the STIFF server array.

9.3 Phase 3: finalizing a minimalistic but scalable architecture

At this point it will already be possible to have an early estimation about the computing power required by the transitional on-demand STIFF system, and eventually invest in one or several other PowerEdge SC1425 units to cope with the future load of the final STIFF system. As said earlier, the final STIFF system will be capable of multithreading and it would then be profitable to buy performance rack servers based on 2 to 4 processors, to get the fastest response time per machine of the cluster.

Also, since it is likely that the STIFF system will need a fast connection to the Oracle database once the product information is moved from ProD to the former, and since the network traffic will significantly increase with the company's growth, we recommend upgrading the network infrastructure to support communications at a speed of at least one gigabit per second (1Gbps).

Cost: about €5000

The new rack servers already come with gigabit network adapters, so the cost will mostly be implied by upgrading existing hardware and cabling, and also centralizing the servers in the same location to benefit from Local Area Network (LAN) connectivity.

9.4 Phase 4: achieving zero-downtime

The idealistic network architecture for SureThing, in order to satisfy performance and reliability needs, would be something like this:

Note that only one box was drawn for the CHIPS system, when actually there can be several CHIPS applications interacting with the UCIS system, but we believe that it is not relevant in this view. Same with the BuRP system.

The load balancers will make it possible to spread the load across a cluster of several computers performing the same task, designated as "nodes" of the cluster. Concretely, thel load balancers will be hardware such as the Coyote Point Equalizer E250i, available for approximately €3000 a piece. There are cheaper load-balancing solutions, but this particular one is capable of distributing the load unevenly between computers of different capabilities, and as an extension of this, is also capable of detecting node failures and immediately stop forwarding connections to affected nodes.

The "standby" load balancer attached to each of the "primary" load balancers is there to provide redundancy in case of failure of the primary, completing the system's insurance of zero-downtime.

The Oracle database is different: for all the other systems the risk is mainly of the system's logic to fail and loss of data on those has no significant consequence, whereas for the database, the data is crucial because it represents company information that has been accumulated in years of business... The risk of software failure on the Oracle server has to be considered minimal, according to its vendor's slogan: "Oracle, the unbreakable database", and anyway it would be quite difficult to provide redundancy for the system at the software level. However, hardware redundancy can easily be achieved by duplicating each hard disk drive using RAID technology, which would provide zero-downtime recovery in case of hard disk drive failure. Furthermore, also making back-ups of the database on tapes and storing those at another location would prevent loss of data in case of major disasters such as fire in server room.

Cost: about €30000

€9000	primary load balancers
€1000	second web server for system redundancy
€1500	second UCIS server for system redundancy
€2000	second STIFF server for system redundancy
€3500	RAID array on Oracle server for hardware redundancy
€3000	tape drive + sufficient set of blank tapes for backups
€9000	extra load balancers for network redundancy

Note that **these are all optional and can be invested in at any time**; the capital budgeting decision in regard to the insurance of zero-downtime is up to SureThing's managers.

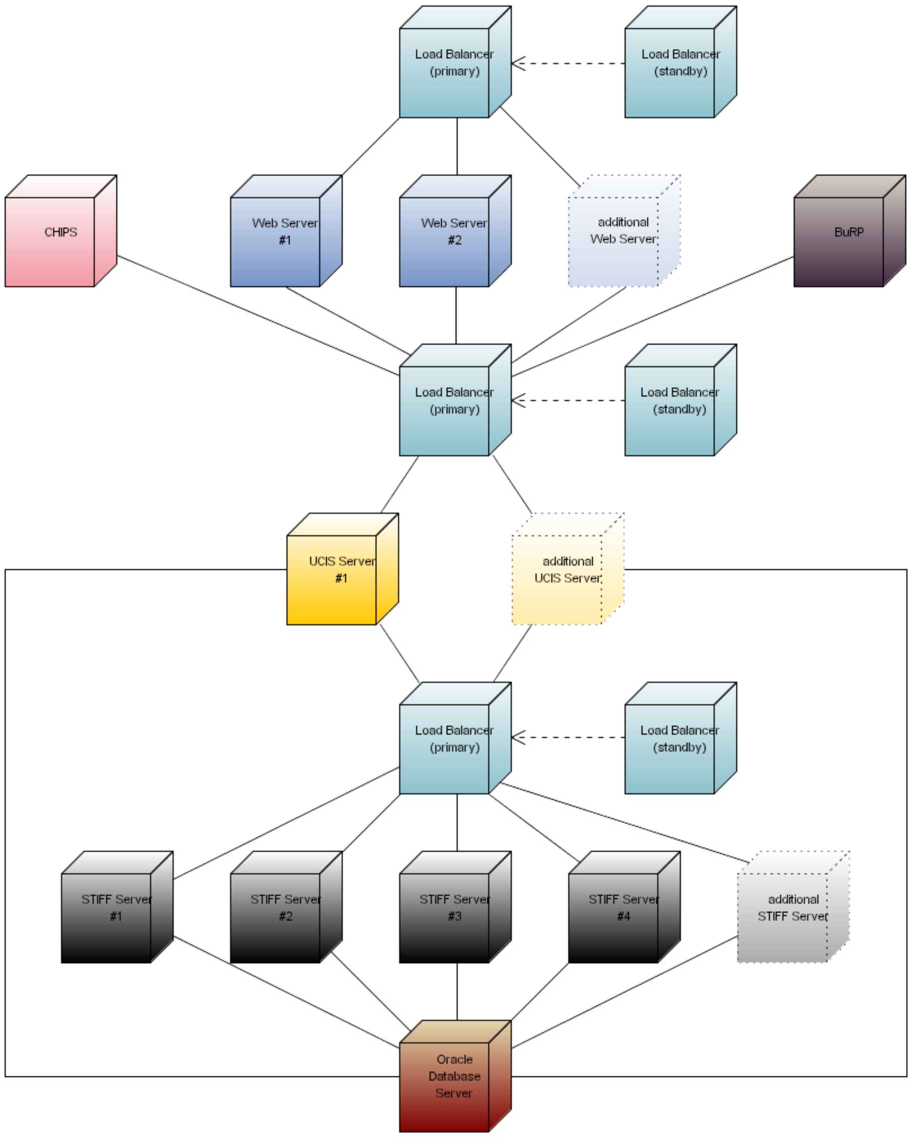


Figure 9.6: Ideal network configuration

Chapter 10

Scenarios View

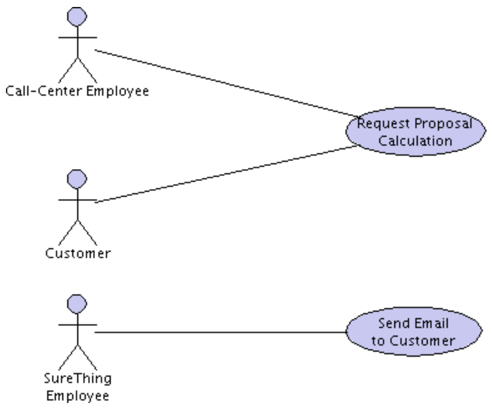


Figure 10.1: Illustration of representative use cases

This view is a guide through the previous four views (Logical, Process, Development and Physical) as it shows how a set of use cases which are representative for the system (see figure 10.1) can be carried out in this architecture.

10.1 Scenario: Request for proposal calculation

A use case which we see as relevant for this system, both in terms of coverage of architectural components and in terms of the frequency with which it appears in real-life is the request issued by either a Call-Center employee or by a customer for a proposal calculation.

This use case is defined as follows:

Use case: Request proposal calculation
Level: User goal
Primary Actor: Call-Center Employee / Customer

1. Customer requests proposal calculation.
2. System gathers data needed for calculation from the customer.
3. System communicates data over the network to the company's central server.
4. Central server gets request for proposal calculation, performs it and:
    (a) stores the result into the database;
    (b) sends the result back to the requesting customer.
5. System running on customer's host displays proposal to the client.

The way this use case's functionality is achieved differs form one evolutionary phase to the next. All views cover the system in each of these phases, making it clear what phase each diagram refers to. Therefore, it is advisable, when working on phase `x`, to read the part describing phase `x` from each of the views.

Analysis can begin with the logical view, which makes clear what components of the system are involved in dealing with this use case. In phase 1, for example, you can see in figure 6.2 that the action is initiated by the GUI and client UCIS component running on the Call-Center employee's computer, then the request goes to the central UCIS server, where it is then forwarded (after some possible processing) to the STIFF system by means of the STIFF Wrapper. Looking at the process view can be particularly useful here in order to understand how the STIFF Wrapper achieves communication with the STIFF system itself and is then able to provide the result back to the UCIS system. Naturally, the result is stored (as specified) in the database, and also sent back to the client UCIS component and displayed on screen.

If you are interested in understanding how the use case is handled in phase 2, you should consult the logical and process views to notice how the UCIS system is now modified in order to allow web-access. The use case now is initiated by means of the customer accessing a certain proposal calculation request web page and making an HTTP request to the central UCIS system. The logical view shows how the client UCIS component disappears and it is replaced by a streamline web browser, while the UCIS server is modified so that it now includes all the previously shared functionality in a form suited for use as a web application. Thus, the central server, which is the one handling the customer's request, is now accessed as Java servlet processing a HTTP request. The diagram from figure 7.6 of the process view shows how the *Front Controller* pattern is employed in order to dispatch the customer's request to a particular component of the UCIS server. The processing from here on proceeds in more or less the same way as in phase 1 and you are therefore referred to the explanation of this phase. In direct connection with this use case is also the physical view, which helps you understand how the system should be set up in order to function in a web environment capable of handling use cases like this one.

Finally, for running the use case in phase 3 (the final development phase), proceed in the same way as for phase 2 up to the point where a request has to be made to the STIFF system and then examine the logical view of phase 3 to understand the changes made to the STIFF system. The final part of the use case implies the actual computation of the proposal, preferably in real-time. Here's where the process view (section 7.3) and the physical view come into play to give you a clearer image of how the STIFF system can be distributed in order to provide real-time response to requests for proposal calculations, even as the number of simultaneous requests increases. The logical view will also show that the ProD has been, by the end of this phase, moved to the Oracle database, so STIFF now can get all the data necessary for its computation directly from the this database.

The development view will not be of particular use for understanding how this use case itself can be realized, but it will be of help to guide development in such a way that the handling of the use case in each evolutionary phase brings an additional performance increase. The development view will explain the rationale behind the decisions to split the development process in the way in which it has been split and it will also explain why certain development steps have been included in a certain phase and not in another one. Thus, the development view will be of interest especially to the developers which have to understand the sequence in which various components that play a part in the realization of this use case have to be modified or extended.

10.2 Scenario: Send email to customer

The second use case which we see as significant for our architecture is tied only to the last phase of development, when the so-called GOC (Generic Output Component) is introduced into the system. This use case deals with a request made by an employee of the SureThing company for an email to be sent to a customer. The contents of the email is irrelevant (it could be anything from an invoice to a claim handling solution). Also, the form of the message (i.e., email) is irrelevant as well as it could be any other of the forms provided by the GOC component (fax, paper and so on). Finally, the addressee (i.e., the customer) is also just an example as it could very well replaced with another employee of the company, for instance. In light of these facts, a more generic name for this use case could perhaps be "Send message to some recipient."

This definition of this use case is provided below:

Use case: Send email to customer
Level: User goal
Primary Actor: Company Employee

1. Employee requests (usu. by means of either the PROFI or the UCIS system) that an email containing some relevant information be sent to a customer. As an alternative, the system could generate a request of this kind itself, as a result of some (e.g.) checks.
2. The system forwards the request, together with the contents of the email, to the GOC component.
3. The system indicates that the requested means of communication is email.
4. The GOC component complies and sends the email to the customer.
5. The GOC component informs the system of the result (success/failure) of the request.
6. The system notifies the employee about the result of its request.

As mentioned already, this use case only relates to the system after the 3$^{\text{rd}}$ phase of development, so you are directed solely to the part of each view which describe this phase. The logical view will show that the only two systems directly connected to the GOC component are the UCIS and the PROFI systems, which means that should a request for output be possible from the CHIPS or BuRP subsystems, it would have to go through UCIS. The development of GOC is beyond the scope of this architectural description (as the component will be provided "as is" by a 3$^{\text{rd}}$ party), and therefore you will not find detailed information about its inner structure herein. What both the process view and the logical view will imply with respect to this use case is that PROFI, as well as UCIS, will have been changed in such a way that they no longer use their own modules for producing output, but rather that they interface with GOC and forward all such requests to it. Consult the physical view if you are interested in the actual hardware on which the GOC will be deployed. This will probably concern you if you are the one installing this component in the system.

Chapter 11

Security View

As one of the main concerns of the acquirer (stakeholder) is for the final system to be as secure as possible, both in terms of information confidentiality and in terms of having increased protection from malicious users, we have decided to offer a separate view of this architecture which concerns itself exactly with these security aspects. Also, this view is not redundant in any way with the other views as it covers (in a very large proportion) matters that are not discussed in any of the other views.

11.1 User management

One of the aspects of security which is of concern to the acquirers is making sure that access to parts or all of the system can be restricted only to authenticated users. In order to cope with this requirement we propose a 2-level user management strategy which we see as fit for this particular case. While deciding upon the particular security model to use, we also took into account the acquirer's concern of limiting the changes to the system (and, consequently, the costs implied thereby) to a manageable level.

The first of these two levels deals with the situation that by the end of phase 2 of development both customers themselves and Call-Center employees can access the functionality of the system through the Internet. However, we have assumed that Call-Center employees (should) have access to an super set of the facilities the customers (should) have access to. This implies that a distinction has to be made between the two kinds of users. Also, we envision that it is possible that in the (perhaps near) future, the system will be upgraded so that even more categories of SureThing's employees than merely Call-Center employees will be able to use it in a web-based manner. Clearly, such categories of users will have to be distinguished as well one from another and also from the simple customers. We have therefore decided to enforce authentication and checking of all actions performed by any users so that we prevent unauthorized users from seeing information (or, even worse, modifying it) they would normally not be allowed to.

In order to achieve this we propose that the structure described in figure 11.1 be created into the Oracle database and, subsequently, write UCIS in such a way that user credentials are checked on login, stored as part of the session and checked every time a request for performing an action is made. Such checks are to be done by the Front Controller servlet (see Logical View) so that:

- the code is not duplicated in each class (so that it is easier to change, easier to test, easier to set up);
- we avoid forgetting to include them in some of the classes.

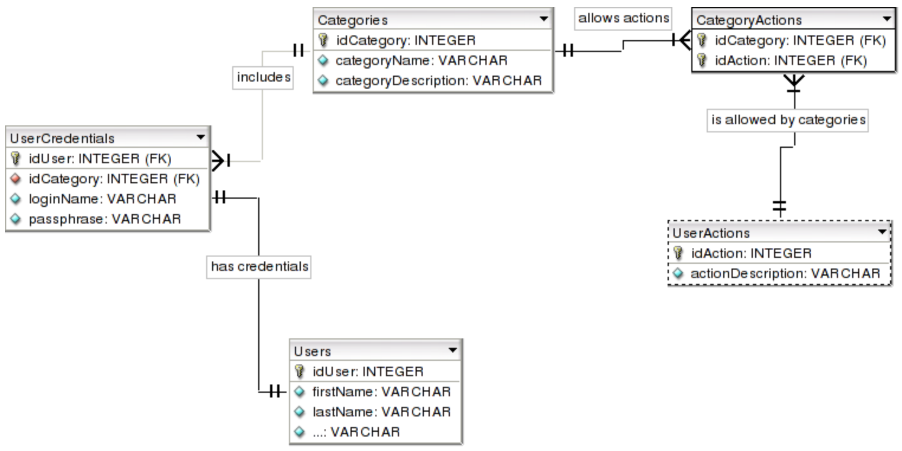


Figure 11.1: Model of the security-related tables. Explanation of notations: rectangles represent tables, with the table name and table fields being written in the rectangle. The line that appears between fields of a table separates fields which are part of the primary key (above the line) from fields which are not part of the primary key (below the line). Fields marked with (FK) represent foreign keys which have been exported from another table. The names of the fields are separated from their respective types by a colon. Lines between rectangles represent relationships between tables. Thin(ner) lines represent non-identifying relationships, while thick(er) lines represent identifying relationships. The symbols at the end of the lines indicate multiplicity. Two parallel lines show a multiplicity of *1*, while a line and a three other converging lines show a multiplicity of *n*. Finally, relationship names should be read like: <table exporting foreign key> <relationship name> <table importing foreign key>.

Security rights should be associated with actions in a database table (as opposed to in a configuration file) - here called CategoryActions - so that scalability is further ensured (i.e. we don't have to worry about having all the copies of the configuration files consistent because all the information will be in just one place, namely in the database). In order to avoid large number of accesses to the database, the Front Controller servlet will read this information on initialization once and use it throughout its lifetime. This means that when the security rights are modified, all the running Front Controller servlets have to be restarted.

The second level of security is the database access security level, i.e. using the database user checks which Oracle allows. A database user is to be created for each category of employee or customer that accesses the IT system and configure subsystems, depending on what category of users use them, to use the particular database user which is associated with the category of employee using the subsystem. As such, PROFI will use, for example, the accountant database user, CHIPS will use, for example, the claim handler database user and so on. This second level of security is introduced to deal with the acquirer's concern of wanting to achieve a certain degree of internal control of access to data (i.e., allow certain categories of employees access only to certain parts of the database). Although not very flexible, we believe that this scheme is appropriate as a first step in achieving internal security. Of course, employees which don't access the system through the Internet could be included in the first level scheme as well, but this would imply a lot of changes to all of the current subsystems (CHIPS, BuRP and PROFI), which the acquirer wants to avoid.

11.2 Protection against malicious users

This second section deals with the acquirer's concern that his customers' confidentiality is ensured, that the internal system is well protected against attacks and that all sensitive information sent over the network is protected from eavesdropping. In order to address these concerns we propose a number of security measures which we recommend that are taken when developing and deploying the system.

To provide customer confidentiality as well as secure authentication for all users which access the company's web site through the Internet, the HTTPS (i.e., HTTP using Secure Sockets Layer) protocol should be used for all such sensitive communication. It is not vital that HTTPS be used for all communication that takes place over the public network, but it is vital that at least authentication of users and communication of confidential customer information use HTTPS.

The company's network should be configured in such a way that none of the company's machines except the web server(s) are accessible from the Internet. This will introduce an additional level of protection in the way of possible attackers, as they would first have to gain access to the machines running the web server(s) and only then could they start attacking internal machines. In order to achieve this, we recommend that the load balancer for the web server that is proposed in the physical view be bought from the very beginning (even if currently the company does not require more than one web server to handle the customer load). As the load balancer is actually an extended router, it can be configured to serve as a firewall which denies access to all ports other than 80 (or whatever port the web server(s) are running on). Sure enough, once the customer load increases and the company buys a second machine to run a web server to deal with it, the load balancer will continue to behave as a firewall as a secondary purpose and also start acting primarily as a load balancer. This suggestion comes as a solution for the acquirer's concern of protecting its internal network from attacks.

Since if the previous suggestion is put into practice the web server becomes the only gate of access to the internal network, two additional measures can be taken for further securization of the company's network:

- use a secure operating system on the machines running the web server (e.g. OpenBSD, NetBSD);
- use a secure web server to run the application (e.g. the Apache web server).

In addition from being secure, such software would also help reduce the costs as any of the above mentioned systems are free.

Another aspect which we have in mind is to also secure the communication over the (current) 2 MBit lines connecting the IT center from the other 4 locations where the company runs its business. In order to achieve this, we recommend configuring the networks in the 5 locations as a large Virtual Private Network (VPN) and make sure that the routers which provide the connections (at all ends) use IPSec for all data communication between the locations. This way, any possible naive eavesdropping can be avoided, while ensuring a certain level of protection against less naive attempts as well.

Last, but not least, we can also suggest an additional measure of protection against attackers gaining access to the most sensitive of the company's data by installing a (software) firewall on the machine running the Oracle database and configuring it in such a way that all attempts of accessing the database made from machines other than the ones running the UCIS system and the STIFF system are disallowed. In this way, even if, say, an attacker gained control of some of the internal machines (other than the ones running UCIS/STIFF), it would have a hard time accessing company data which is stored in the Oracle database.

Chapter 12

Conclusion

Using the directives stated by this architecture description, SureThing will benefit from an IT system that is completely scalable (ready to cope with the company's growth expected after the merger) and unbounded (very unlikely to be threatened by the eventual discontinuation of proprietary technologies). In four words: SureThing will be **ready for the future**.

12.1 Why the .NET requirement has been discarded

The .NET framework is a new technology released by Microsoft in order to compete with Java, as-a-matter-of-factly coming with its share of advantages and disadvantages.

On the one hand, programs in .NET are supposedly slightly faster than the ones written in Java, but not significantly: both use run-time emulation via a virtual machine, which makes them about two to three times slower than native programs (programs that have been translated to machine code).

On the other hand, this performance overhead is forgivable for Java, because this separation between virtual machine code and machine code makes it possible to run the very same Java program on a vast variety of platforms whose manufacturers have already agreed to comply with Java standards, whereas .NET's expansion to other platforms than Microsoft Windows does not seem promising at all.

Therefore, had we chosen to develop the SureThing architecture using the .NET framework, no savings could have been made by reusing part of the existing J2EE implementation of UCIS. Instead, the project would have suffered incremental costs implied by the licensing of the .NET technology. As for the future, it is impossible to assert which of the two rival techologies will prevail, but one thing is for sure: today, the Java technology is widely spread whilst the .NET technology has yet to prove itself.

Finally, considering the fact that the future of the Microsoft Windows operating system itself has never been less sure since it overcame Apple's MacOS on the market in the late eighties / early nineties, with the Apple's recent comeback stories on every business channel nowadays, and worldwide organizations such as the UN praising the Linux operating system to the people, and computer market leaders such as IBM, HP and SUN promoting Linux-based IT solutions, and the increasing security threat to users of the Windows operating system, we believe that chosing the Java technology will bring extra insurance to the project in regard to the uncertain future of computer operating systems and architectures, since it is possible to easily move Java applications from a Windows- and PC-based platform to any other platform supporting Java, which is approximately *any existing platform*.

12.2 Why the buy-before-build strategy has been ignored

The project is mainly about merging two existing architectures and enhancing the result. No software on the market would be exactly suitable for the task, and buying a whole new software system would prove to be costlier than building on top of the existing system as described by this proposal.

Bibliography

[Fowler2000] Fowler, M.: UML Distilled Second Edition - A Brief Guide to the Standard Object Modeling Language. Addisson-Wesley, 2000.

[Fowler2003] Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley, 2003.

[Kruchten1995] Kruchten, P.: Architectural Blueprints - The “4+1” View Model of Software Architecture. *Published in IEEE Software 12 (6)*, Nov 1995, pp. 42-50.

[cron] CRON Manual Page *Online http://www.rt.com/man/cron.8.html*, January 2005.