Gliwice 2001-09-25

SILESIAN UNIVERSITY OF TECHNOLOGY
FACULTY OF AUTOMATIC CONTROL, ELECTRONICS AND COMPUTER SCIENCE

MSc thesis

Design and Implementation of an XML-based Call-Tracking-System for Monitoring and Improving Business-Process Performance

Supervisor: **PhD Henryk Małysiak**
Consultant: **PhD Lars Hagge**

Author:
Oskar Werewka

Table of Content

Summary in Polish Language

Spis treści:

Wprowadzenie

Deutsche Elektronen Synchrotron (DESY) w Hamburgu jest instytutem naukowym prowadzącym badania w dziedzinie fizyki wysokich energii. W planach instytutu znajduje się budowa nowego akceleratora liniowego długości trzydziestu kilometrów o nazwie TESLA. Projekt ten został zaprezentowany niemieckiemu rządowi. Obecnie oczekuje na wyrażenie decyzji o finansowaniu.
W trakcie przygotowań do projektu TESLA w 1999 roku wydział DESY odpowiedzialny głównie za IT, inżynierię elektroniczną, mechaniczną oraz zachowanie norm bezpieczeństwa zdecydował o utworzeniu nowej grupy o nazwie Information Management Process, Projects w skrócie IPP. Głównym zadaniem nowopowstałej IPP jest wprowadzanie nowych strategii w projektowaniu urządzeń mechanicznych, tak by proces ten był efektywnie wspierany za pomocą narzędzi CAD.
Zidentyfikowano następujące procesy, które będą stanowić główną część usług udostępnionych przez IPP dla innych grup w ramach instytutu:

1. Przygotowywanie stanowisk CAD.
2. Szkolenia użytkowników.
3. Pomoc techniczna.
4. Konserwacja stanowisk.
5. Dostarczanie dokumentacji oraz informacji bieżących.

W trosce o zapewnienie odpowiedniej jakości oraz efektywności wymienionych powyżej usług oraz dla uzyskania bieżącej informacji o będących w toku procesach specjalny system komputerowy powinien zostać wykonany i wdrożony w IPP. System ten powinien zbierać dane dotyczące każdego wniosku o wykonanie określonej usługi, oraz w miarę postępów w pracy zachowywać jej historię.

Cel i przyczyny:

Jest wiele czynników świadczących o tym, że wprowadzenie nowego systemu odbije się pozytywnie na jakości usług dostarczanych przez IPP. Komunikacja wewnątrz grupy oraz pomiędzy grupą a jej klientami jest w znacznej mierze oparta na poczcie elektronicznej. Jest ona dość wolna i może stać się nużąca, kiedy większa ilość pracowników weźmie udział w rozwiązywaniu określonego problemu. Dlatego potrzebny jest system, który stanie się

centralnym miejscem zarządzania informacją, w którym nastąpi integracja komunikacji jak i funkcji dokumentujących. Bez wsparcia systemu, osoba, dla której IPP uruchomiła proces musi pytać odpowiedzialnych za niego pracowników telefonicznie, jaki jest jego stan zaawansowania. W optymalnym przypadku osoba ta mogłaby samodzielnie wydobyć tą informację z systemu.
Kierownik IPP potrzebuje zestawu statystyk na podstawie, których może on podejmować decyzje. Statystyki są pomocne w identyfikacji powtarzających się błędów w realizacji procesów, umożliwiają one ich uniknięcie oraz bardziej precyzyjne planowanie przyszłych inwestycji. Obecnie wykonywanie statystyk w IPP jest raczej skomplikowanym zadaniem, które w dodatku może wykonać tylko specjalnie w tym celu oddelegowany pracownik. Dokładność sporządzonych dokumentów i tak jest ograniczona, ponieważ po zakończeniu danego procesu większość detali związanych z jego realizacją nie jest nigdzie dokumentowana.

Oczekiwane korzyści z realizacji projektu:

Operator linii pomocy technicznej:

Osoba ta zachowa więcej czasu, ponieważ będzie rejestrować informacje tylko w jednym miejscu, nie będzie musiała dzwonić albo wysyłać poczty elektronicznej do swoich współpracowników aby ich powiadomić o rozpoczęciu nowego procesu. W dodatku operator jest odpowiedzialny za kontrolę postępów w pracy, musi sprawdzać on czy nie istnieją zaległości w realizacji zadań. Kontrola ta będzie znacznie ułatwiona dzięki informacjom dostarczanym przez system, nie trzeba będzie telefonować do określonego pracownika by dowiedzieć się, co zostało zrobione.

Klienci IPP:

Do dyspozycji klientów zostanie oddana specjalna aplikacja dostępna w intranecie. Za jej pomocą będą oni w stanie sami rejestrować informacje w systemie z pominięciem linii pomocy technicznej. W przyszłości będzie to standardowy sposób rejestracji żądań kierowanych do IPP. Osiągnięty zostanie znaczny stopień automatyzacji. Klienci także łatwo będą mogli sprawdzać stopień zaawansowana procesów ich dotyczących. Konieczność telefonowania czy pisania listów elektronicznych z pytaniami zostanie wyeliminowana, w dodatku system będzie dostępny przez całą dobę.

Pracownicy IPP odpowiedzialni za wykonywanie procesów:

Dla każdego pracownika zostanie dynamicznie utworzona lista zadań do wykonania. Lista ta będzie zawierać informacje o wszystkich procesach, w których dany pracownik musi wykonać jakąś część pracy. Informacja ta będzie prezentowana w sposób zwarty i w jednym miejscu. Pracownicy będą w stanie określić swoje obciążenie pracą w kilka sekund, dzięki czemu będą lepiej zorganizowani.

Kierownik grupy:

Zbiór statystyk będzie generowany w sposób automatyczny i w krótkim czasie. Ponadto statystyki te będą się odznaczać dobrą dokładnością wynikającą

z relatywnie dużej ilości danych gromadzonych przez system. Posiadając je łatwiej będzie dostosować IPP do stojących przed nią wymagań.

Sposób realizacji:

System został wykonany z użyciem iteracyjnej metody konstrukcji oprogramowania. W metodzie tej czas przeznaczony na realizację projektu jest podzielony na tak zwane iteracje, o czasie trwania określonym w tym przypadku na sześć tygodni. Skład pojedynczego cyklu jest następujący:

- Początkowo pierwsze dwa tygodnie zostaną spędzone na zbieraniu wymagań użytkowników.
- W następnym tygodniu przeprowadzona będzie analiza zebranych wymagań. Na jej podstawie zostanie wykonany projekt architektury systemu. Jest to pierwsza faza metody iteracyjnej. Architektura będzie w kolejnych cyklach ewoluować by zaadoptować zmiany jak i pojawiające się nowe wymagania.
- Kolejne dwa tygodnie będą przeznaczone na implementację. Zostanie ona przeprowadzona tylko w wyznaczonym zakresie. W następnych iteracjach zakres ten będzie się rozszerzać.
- W następnym tygodniu będzie wykonana weryfikacja techniczna oraz użytkowników. Będzie przeprowadzone testowanie, dyskusje oraz analiza spełnienia postawionych wymagań.
- Jeden tydzień jest przeznaczony na udokumentowanie bieżącej iteracji.
- W ostatnim tygodniu zostaną zebrane nowe wymagania użytkowników, co zamyka cykl. Jeśli będzie to konieczne następny zostanie rozpoczęty.

Iteracje powinny być powtarzane do osiągnięcia odpowiedniego poziomu funkcjonalności i niezawodności. Sześciotygodniowa jej długość będzie pozwalać na łatwiejszą konserwacje, oraz dobre udokumentowanie systemu. Graficzne przedstawienie cyklu jest pokazane na rysunku numer 1-1 (Fig. 1-1).

1 Introduction

DESY in Hamburg is a research institute for high-energy physics and application of synchrotron radiation. The institute is planning to build a new thirty kilometre long linear collider, called TESLA. The TESLA project is currently proposed to the German government and is waiting for approval.
In preparation for the TESLA project in 1999, DESY's division for central services has performed a business reengineering. The central division includes the IT services, mechanical and electronic engineering and technical safety groups. As a consequence of the business reengineering effort, a new group has been established to develop and introduce new strategies for mechanical engineering to support the CAx systems. The group is called Information Management Process, Projects (IPP).
IPP has designed its business processes and identified the following major services for the CAD support:

1. CAD seat preparation
2. user training
3. user support/hotline
4. system maintenance
5. information and documentation

To measure the efficiency and quality of these services, and to obtain an up-to date documentation of ongoing processes, a process monitoring system shall be developed and introduced in IPP. The monitoring system shall create a "ticket", i.e. an information record for each service request, and log the history as the service request is worked upon. This thesis describes the development and implementation of a Trouble Ticketing System (TTS).

1.1 Goal and purpose

The main goal is to build a Help Desk system to support IPP group in coordination and documentation of ongoing work processes. The system should be helpful with maintaining and storing activity information. The TTS will provide both internal and external users (further called clients) with the most recent information in an easy way. It will also reduce the amount of mutual cooperators communication, leaving them more time to their own responsibilities.

There are several purposes for introducing a new system. The communication within service group and between service and service customers is currently based on

phone and E-mail. This communication is relatively slow and can become tedious when many people are involved. Therefore a system is needed which could be a central point of information management to integrate communication and documentation. As there is no such a central place where the information is stored, clients will have to ask workers what they are doing when they want to know the state of each process and each activity running. The system will instantly provide this information without additional effort using an appropriate interface. The clients could check what is going on whenever they want to, without needing to contact any IPP worker. Also the supervisor needs to have statistics. Base on them he can improve service or make more accurate planning. For now collecting data for such a statistics is difficult, because usually after solving a problem the majority of details is simply forgotten. In addition, to make a statistic a human being is needed. After introducing TTS, all data will be collected automatically, and statistics will be generated whenever needed without any interaction.

1.2 Overview

First, the scope of the work has to be defined; the most important point is to understand ongoing processes and their lifecycles. With this knowledge it will be possible to find out who will be able to use the system and who will be able to take advantage of using it. It is recommended to talk to each worker in IPP group who will probably use this system in the future. During this interviews workers will be asked what is the wanted functionality (that will make work easier or faster), what are their opinions, proposals, and how they are imaging working with the system. The result of this stage will be users requirements database.

Second, the use cases have to be identified. Than user requirements analysis has to be done, including dividing all requirements into three groups: the most important functions that consists the main system functionality, less important features useful but not absolutely necessary, and the least useful. There is done a prototype system, which has to be simultaneously evaluated through realized use cases.

Having requirements and use cases database opens way to design system architecture, which can implement them and is scalable, flexible, robust for changes and maintainable. Next, implementation technologies have to be chosen, and then the implementation, user trainings and documentation have to be done.

The expected results of the project were:

- Central database where the information about ongoing and finished processes will be stored. Each process will consist of a subset of activities. Both processes and activities will have a property that will indicate current state. There will be option to configure the type all used process tickets and activities, so the TTS will not be bound, to certain set of processes but will be able to follow any well-defined process.
- An application, which will give access to information, stored in the central database through any workstation in DESY. Because some portions of this information maybe confidential, there will be some security mechanism which will give full access only to registered users. The clients will have unprotected

access to a chosen subset of information on purpose to relieve the hot line of calls.

- For registered users who are members of IPP, the application will provide functions to:
    1. To optimize time spent to get information out of TTS, an intuitive representation mechanism will be provided. Workers needs to know which task are to perform; this information will be organized like a to-do list, which will be represented as a tree of process tickets.
    2. Interface designed on purpose to enable hot-line operator to register new problems as they appears.
    3. Generating statistics for planning and controlling.
- Supporting email subsystem. It will be enabled on a user request, and will automatically provide the most recent information. For clients it means that when status of process ticket or an activity change, there will be an instant message. That allows clients to be up to date at lowest cost.

The expected benefits of the project were:

Hot-line operator:

The hot-line will save time, because all needed information will be typed into one place, there will be no need to call or to send special emails to any cooperators to inform them that they have a new tasks to perform. There is also another benefit, because hot-line operator is responsible of controlling work progress, he has to check if there are some activities which should be done and are not. Now it could be done without asking adequate worker what progress has been done.

Client:

Clients will be able to report their problems not only using hot line or e-mail, but also web browser. This gives a standard way to report problems, which relieves the hot line. In the future when clients got accustomed to this way of reporting, the whole process could be well automated. Also whenever wanted to know what the progress in a certain problem is, the client can just ask TTS and receive all wished information.

Process worker:

The IPP worker who has a set of his responsibilities will have a to-do list. That list will include all activities and process tickets aimed only for him. He will have pointed out all new tasks. Thanks to it the person could assess the workload with one glance and optimize the realization.

Supervisor:

The IPP manager will have all needed statistics. They will be made automatically and more precisely, because the system will have accurate data. Based on this it will be easier to plan future activities. Also no effort will be needed to gather them.

TTS will be built using iterative implementation method, which should consist of six weeklong cycles:

- Initially during first two weeks user requirements will be collected.
- The next one-week will be taken by analyzing requirements and on that base architecture will be designed. This is the first phase of the iterative method. The architecture will grow to accommodate changes and new requirements.
- During next two weeks the implementation will be performed. But only in the chosen scope. During following iterations the scope will extend.
- In the next week implemented changes will be verified technically and with users, including testing, discussions and fulfillment of requirements.
- During one week the current iteration step will be documented.
- Next week will be taken by upgrading user requirements. And the cycle will start again from the beginning.

The cycle should be repeated as long as will be needed to reach sufficient functionality and quality level. The six-week intervals enable good capacity for future upgrades to the system.

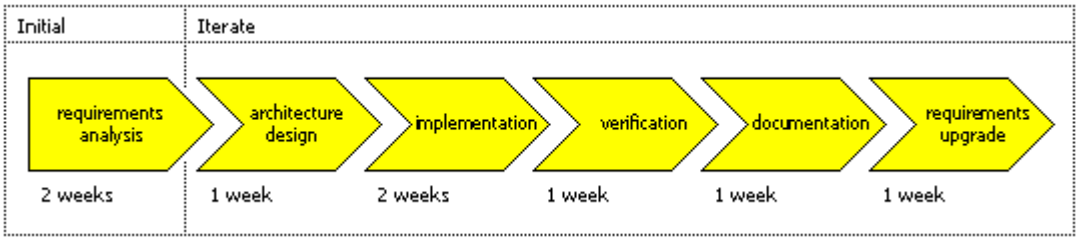


Fig. 1-1 Iterative method development life cycle.

There are several risks involved. Sometimes building a process management system could result in possibly too strict and too complex system, which does not give user any possibility to manipulate processes in a way, which was not foresaw. Such a system could not help to work faster, instead requires typing more not necessary information and irritates users. From DESY's point of view this thesis does not have the only one goal of producing a useful system. There is a requirement to use some new technologies, which should be introduced in the future in DESY IT departments. Therefore some experience in those technologies is needed and it is the reason why they should be used in this thesis. Lack of experience could result in poor system architecture. Because not all technologies are now standardized or are actually changing that could cause problems with scalability and changes introducing.

1.3 Definitions

TTS

Is an acronym for Trouble Ticketing System.

Process Ticket

The term process ticket was created on the basis of definitions of technical and business process:

Technical Process:

A technical process is an operation, which creates, transforms or consumes energy, matter or information. Technical process is shown in Fig. 1-2.

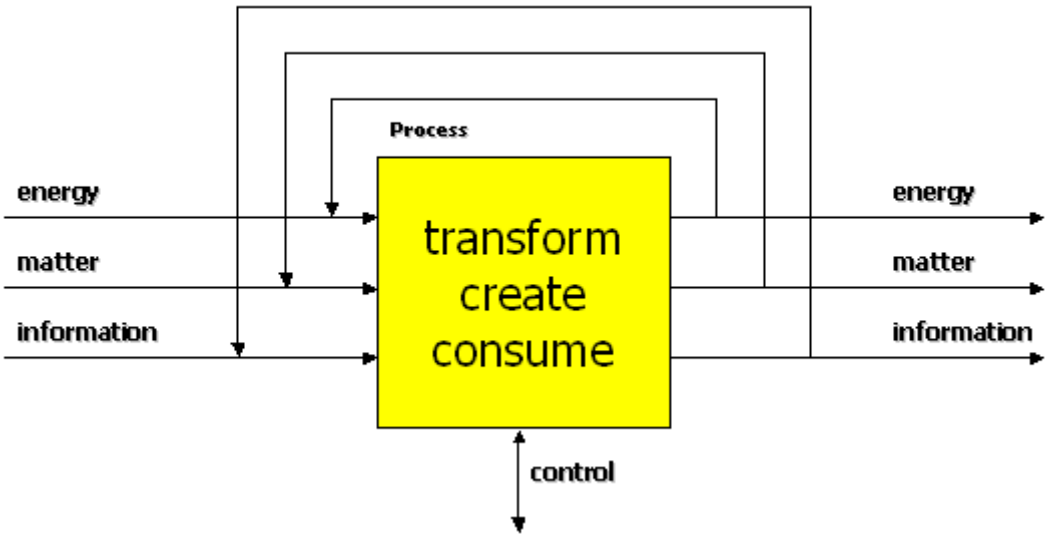


Fig. 1-2 Technical process.

Business Process:

A business process is a set of activities, which are directed to the same goal, provide use to a customer and have well defined input and output.

Process Ticket:

A process ticket is information record for an instantiated (i.e. running, executed) business process which contains a list of actions necessary to finish the process, and which reward the process history.

Activity

All process tickets in TTS will consist of a set of activities. A process ticket will be considered as closed when all or a subset of its activities will be done. An action is taken by a domain expert, and is desired to solve a part of problem in a certain technical scope.

User

Is a person, who is given the rights to access TTS, and is usually a member of IPP group. This could be a domain expert, hotline operator or supervisor.

Client

Are DESY employees or guests who issue service request to IPP. They are not registered as users and access only public parts of the system.

Domain Expert

Possess knowledge in some special areas. They are responsible for solving problems in these areas. For example products related tasks.

Hotline Operator

Is a member of IPP who is responsible for collecting information from clients, informing them about the work progress and dispatching process tickets among domain experts.

2 Requirements analysis

2.1 Introduction

Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for a large building. Good models are essential for communication among project teams and to assure architectural soundness. The models for complex system are built because it is impossible to comprehend any such system in its entirety. As the complexity of systems increase, so does the importance of good modelling techniques. There are many additional factors of a project's success, but having a rigorous modelling language standard is one essential factor.

Goal: Govern changing and evolving system specification

The goal of collecting and analysing requirements is to gain all information needed to first build the system model and then the system itself. In the early stages the collected requirements are not very precise. After analysing the first portion of them, more detailed interviews can be done. Usually after showing to users the first prototype requirements are growing, it is because users can easily imagine what else the system can do for them. It is important to early identify the most important features of the system, and to separate them from these requirements, which are rather fancy wishes. This separation allows concentrating on the most important elements.

Method: "How-to"

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software. It looks for techniques to improve quality and reduce cost and time-to-market. These techniques include component technology, visual programming, patterns and frameworks. The industry also seeks techniques to manage the complexity of systems as they increase in scope and scale. In particular, there are needs to solve recurring architectural problems, such as physical distribution, concurrency, replication, security, load balancing, and fault tolerance. Development for the worldwide web makes some things simpler, but exacerbates these architectural problems.

The Unified Modelling Language (UML) is a general-purpose visual modelling language that is designed to specify, visualize, construct and document the artefacts of a software system. The UML is simple and powerful. The language is based on a small number of core concepts that most object-oriented developers can easily learn and apply. The core concepts can be combined and extended so that expert object modellers can define large and complex systems across a wide range of domains. The UML is the visual modelling language of choice for building object-oriented and component-based systems.

The primary goals of the UML are as follows:

- Provide users with a ready-to-use, expressive visual modelling language to develop and exchange meaningful models.
- Furnish extensibility and specialization mechanisms to extend the core concepts.
- Support specification that is independent of particular programming languages and development processes.
- Encourage the growth of the object tools market.
- Support higher-level development concepts such as components, collaborations, frameworks and patterns.
- Integrate best practices.

TTS is not a huge system; there are use case and sequence diagrams.

A use case diagram is a graph of actors, a set of use cases, possibly some interfaces, and the relationships between these elements. The relationships are associations between the actors and the use cases, generalizations between the actors, and generalizations, extend, and includes among the use cases. The use cases may optionally be enclosed by a rectangle that represents the boundary of the containing system. In fig. 2-1 an example of use case diagram is shown.

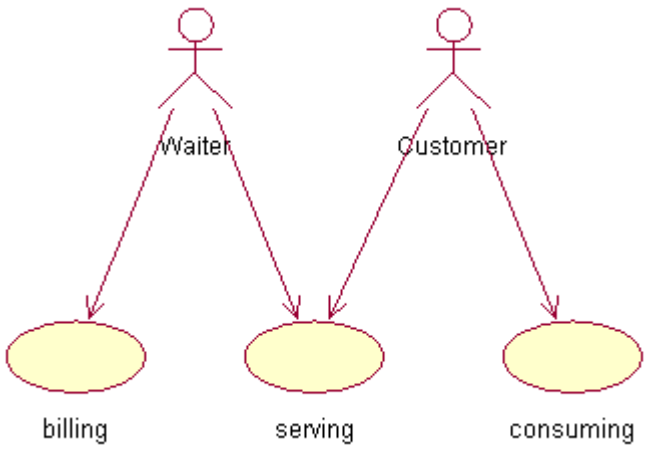


Fig. 2-1 Restaurant use case diagram.

A sequence diagram presents an interaction, which is s set of messages between objects or classes. A sequence diagram has to dimensions:

- The vertical dimension represents time.
- The horizontal dimension represents different objects.

The horizontal ordering of the lifelines is arbitrary. Often call arrows are arranged to proceed in one direction across the page; however, this is not always possible and the ordering does not convey information. In fig. 2-2 an example of sequence diagram is shown.

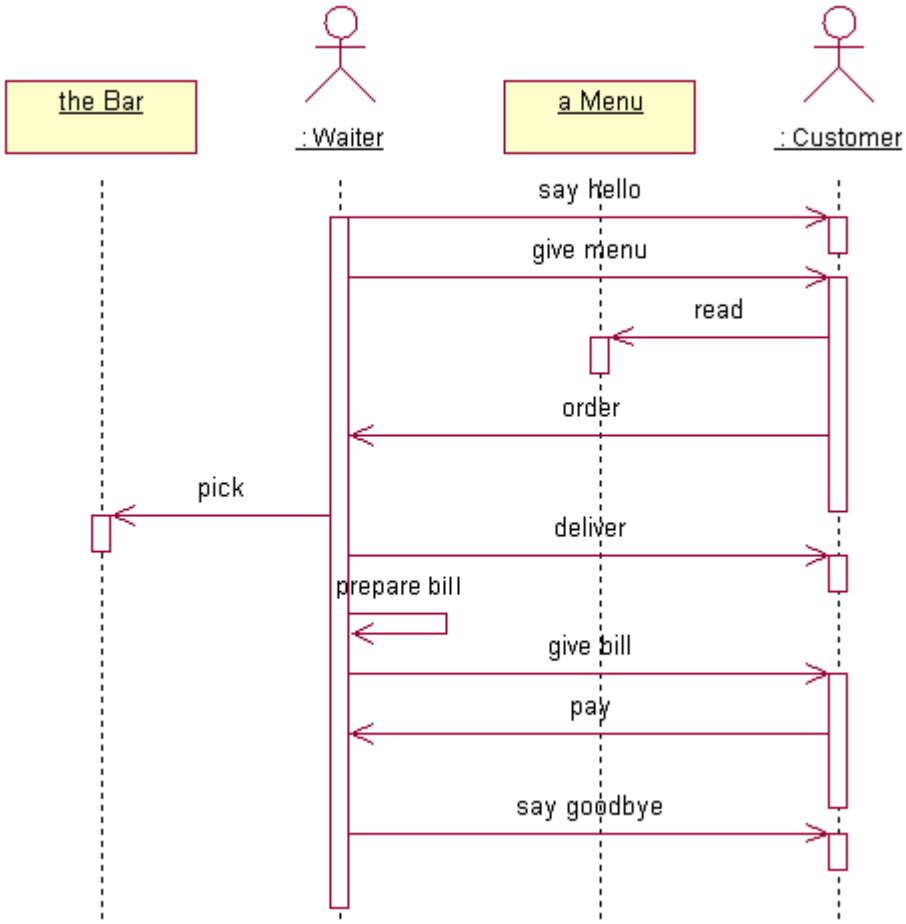


Fig. 2-2 Restaurant service sequence diagram.

2.2 TTS requirements at DESY

Use Case identification starts when user requirements are collected. Use Cases are helpful in many places during system design. The most important feature is that they are useful to understand system behaviour better. They support communication between workers responsible for design system architecture and developers. They can be used for drawing diagrams, which can be shown to visualize some aspects of the system.

The basis to find use cases is a requirements database. Depending of the system size it can be organized in different ways. Because TTS is rather a small system all requirements can be grouped in one two pages long table.

There are defined five features of a single requirement:

- Priority, tells how important a requirement is. Sometimes is could be difficult to find out the real value of this parameter because users likes to tell that everything is very important.
- Difficulty, estimates how it will be difficult to fulfil a requirement.
- Stability, tells if a requirement changes over time. It can happen that people responsible for giving requirements in an organisation will change something every week.
- Clarity, when it is difficult to say what exactly does meant to realize a requirement then is has low clarity.
- Visibility, tells if realisation of this requirement will be visible to the users, some internal algorithms in the system can be completely hidden from users.

Requirements of a high priority or high visibility and high difficulty have big risk. Such a requirements should be implemented first. If they will be left to implement later in another cycle and the implementation will not be successful it could be dangerous to the whole system realisation, users will quickly get to know that there are problems or something is not working as it should be. If requirements of high visibility and low difficulty will be implemented first, this could make users accustomed that they can quickly get something and in the later phase when difficult things of low visibility will be implemented they will ask why nothing happens, or why the development process is now so slow.

In Fig. 2-3 is a list of all user requirements for TTS. It was possible to make an interview with each TTS user, there is a column called “Origin”, which specifies the role of the user who gave the requirement.

Requirement					
Priority	**Difficulty**	**Stability**	**Clarity**	**Visibility**	**Origin**
Filtered process tickets list, where I am the owner of the open process ticket, or I am the owner of at least one open activity.					
High	Middle	High	High	High	Domain Exp.
Process Ticket should have a short remark (subject) one line field, where is possible to type information which uniquely identifies the process ticket.					
Low	Low	High	High	High	Domain Exp.
We should be able to configure for each process ticket a set of allowed activities and statuses; it should be possible that one process have different activities set from others. Also we should be able to configure a set of statuses for each activity, but each activity can have a default statuses.					

Requirement					
Priority	**Difficulty**	**Stability**	**Clarity**	**Visibility**	**Origin**
High	Middle	High	High	Low	Domain Exp.
There should be specified person responsible for adding new keywords connected to the process tickets, that person should also make the keywords comprehensive to other users.					
Low	Low	Middle	High	Low	Hotline Op
Quicker access for user to the new process tickets or activities, when user opens the HelpDesk and log in, if it is possible the new ticket or activity should be in a different color unless the user reads it.					
Low	Middle	High	High	High	Hotline Op
It will be good when there will be a kind of remember, which can inform you when one process ticket or activity is not closed within a certain amount of time.					
Middle	Low	Low	Low	Low	Hotline Op
It can help when we can leave HelpDesk, and it will automatically refresh it, showing the newest information concerning the logged user.					
Low	Low	High	High	High	Domain Exp
We should have a current number for process activities, it will inform us of the order of activities, also for an activity we should have a standard field which point that certain process ticket should have this activity by default. Also standard activities should have a default sequence number.					
High	Middle	High	High	Middle	Domain Exp
It will be good when there will be a main page which by default will contain open process tickets, this page should have very simple access to the place where it is possible to add new process tickets, because it is not needed in case when the user calls us witch his problem.					
Middle	Middle	High	High	High	Domain Exp
One user should have a possibility to view process tickets and activities belonging to others users, and take actions to them.					
High	Low	High	High	Low	Domain Exp
The system should have a multi language support.					
Low	High	High	High	High	Managemnt
There should he a mechanism for informing clients that there are changes made to their process tickets. This should be done by mailing them specific information.					
Low	Middle	High	High	Low	Managemnt
Clients should have a limited access to the TTS, only for searching their process tickets and reading its status and activities information.					
High	Low	High	Middle	High	Managemnt
If somebody will search TTS the information showed to him should not be editable. If he will introduce changes, he should use a special button which will open a page designed on this purpose.					
Low	Low	High	High	High	Managemnt
The users should be able to view all process tickets at once, without choosing a special type of them.					
High	Low	High	High	High	Managemnt
Users should have possibility to specify duration of working time on activity.					
Low	Low	Middle	High	Middle	Managemnt

Requirement					
Priority	**Difficulty**	**Stability**	**Clarity**	**Visibility**	**Origin**
It should be possible to configure in a free manner the following objects: process types, process statuses, activity types, activity statuses, keywords, programs, computers, clients, client types, and notification types.					
High	High	High	High	Middle	Managemnt
Is should be possible to add a new foreign language to the HelpDesk system.					
Low	High	Middle	High	High	Managemnt
When a new process ticket is added, it should be created with his default set of activities, and the activities should have their specified order.					
High	Low	High	High	Middle	Managemnt
For each process ticket users should be able to add new activities from a set for this process ticket type, and specify: start time (system should provide current value), status (from allowed set of statuses), responsible worker, description and if needed extern worker and/or extern call number.					
High	Low	High	High	High	Managemnt
Users should be able to change process ticket data especially including status.					
High	Low	High	High	High	Managemnt
Users should be able to delete, close or change any data connected witch process activities.					
High	Low	High	High	High	Managemnt
Users should be able to view and modify filtered process tickets according to: worker, client, computer, programs, keyword, status, start time scope, subject and description.					
High	Middle	High	High	High	Managemnt

Fig. 2-3 Collected user requirements for TTS.

2.3 TTS system main Use Cases (capabilities)

2.3.1 System structure

TTS interacts with three actors; they are: a *client* who has a problem to solve, a *hotline operator* who registers process tickets, and a *domain expert* who actually solves the problem. There are to main use cases: *provide service*, which services a client and *proceed tasks* for the domain experts. *Provide service* includes three others sub use cases: *request service* for clients, *notify client* to keep them up to date and *dispatch ticket* to distribute activities among domain experts. The *proceed tasks* use case includes two sub use cases which are: *search tasks* to tell domain expert what has to be done and *update ticket* where domain expert changes activities or process tickets to mirror work progress. The TTS use case main diagram is in fig. 2-4.

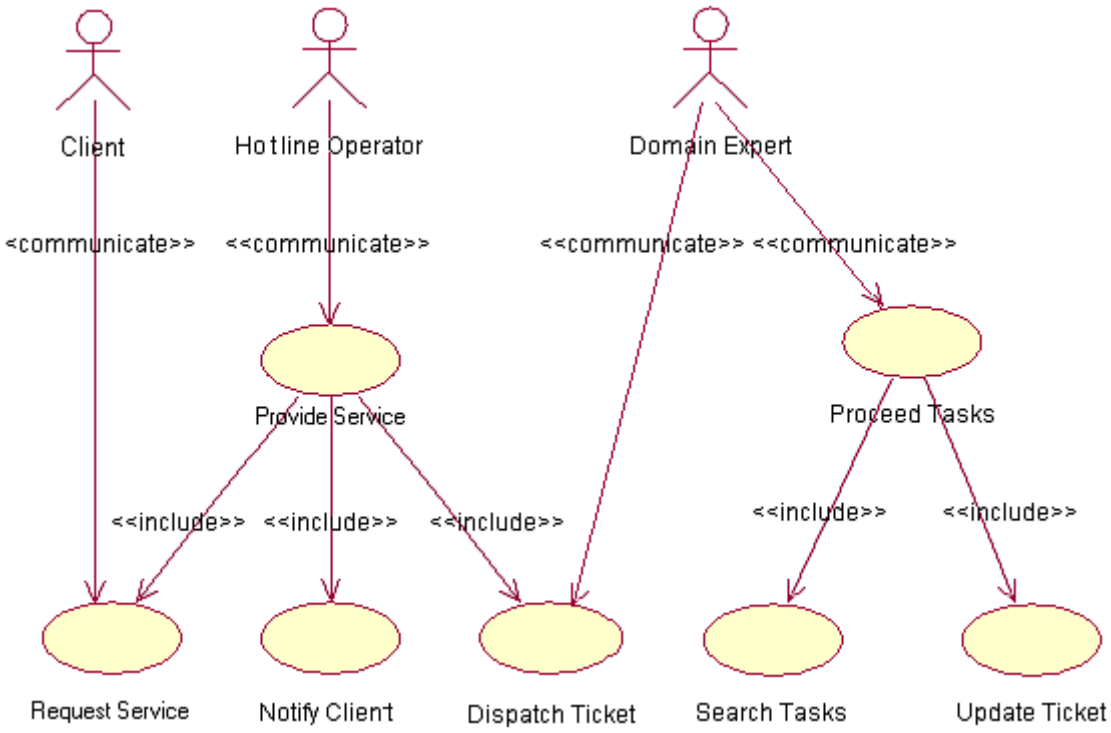


Fig. 2-4 Main TTS Use Case diagram.

To better visualize the general TTS operational model two high level sequence diagrams in Fig. 2-5 and Fig. 2-6 are shown. The first diagram applies to situation when the hotline operator mediates between client and TTS to register a process ticket. In second diagram an unaided client registers ticket. This variant has an advantage of reliving hotline operator also registering process tickets will be possible around the clock.

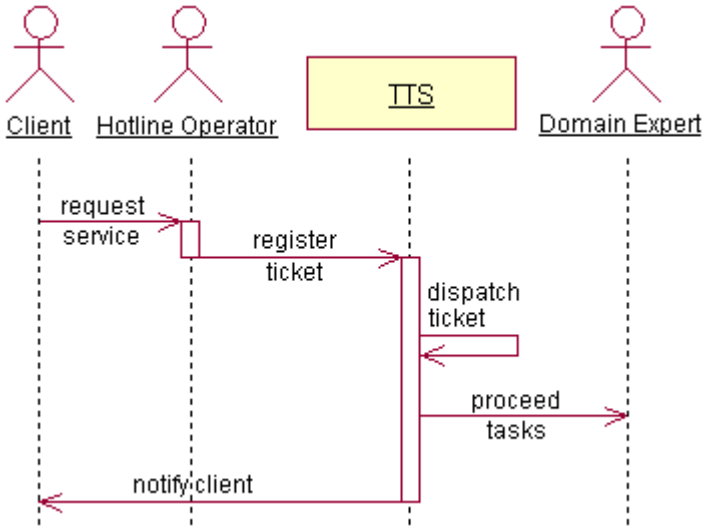


Fig. 2-5 High-level TTS sequence diagram.

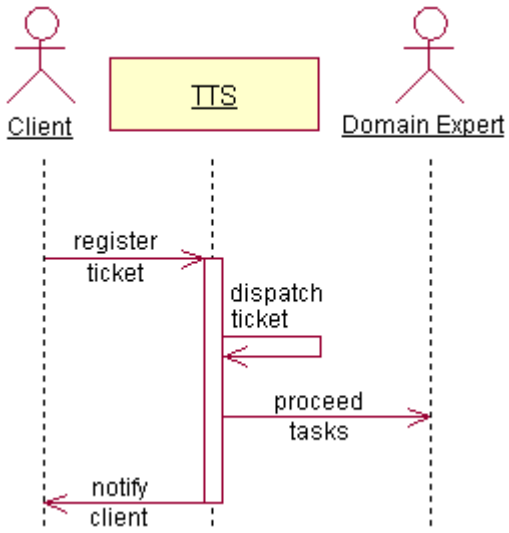


Fig. 2-6 High-level TTS sequence diagram version without hotline operator.

2.3.2 Provide Service use case

This use case starts when a client contacts IPP group and gives description of a problem. Use case ends when a problem is solved.

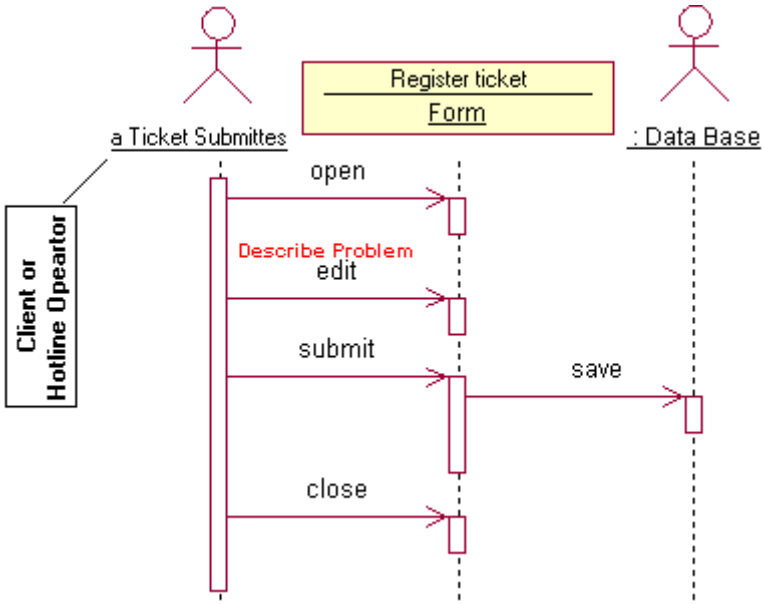


Fig. 2-7 Request service scenario.

Request Service use case starts when a clients calls hotline operator or opens an appropriate form and gives description of a problem, which has to be solved. Use case ends when this description is approved and saved into the TTS. Sequence diagram is shown in Fig. 2-7.

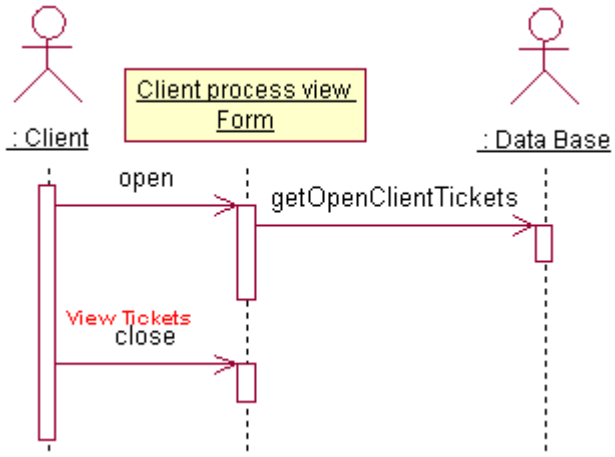


Fig. 2-8 Notify client scenario.

Notify Client use case starts in two cases. When there is a change made to a process ticket. Or when a client opens a special form where he can read status information about a process ticket. Use case ends in first case when an email to a client is send containing status information about changes in a ticket. In second case it ends when client closes the form. Sequence diagram for form variant is in Fig. 2-8.

Dispatch Ticket use case starts when new process ticket is under registration process. The hotline operator assigns the domain experts responsible for this new ticket and its activities. Use case ends when hotline operator submits the ticket.

2.3.3 Proceed Tasks use case

This use case starts when a domain expert will begin work and needs to get information about his responsibilities. During work the information concerning process tickets and activities is changed to reflect work progress. Use case ends when the expert finishes work.

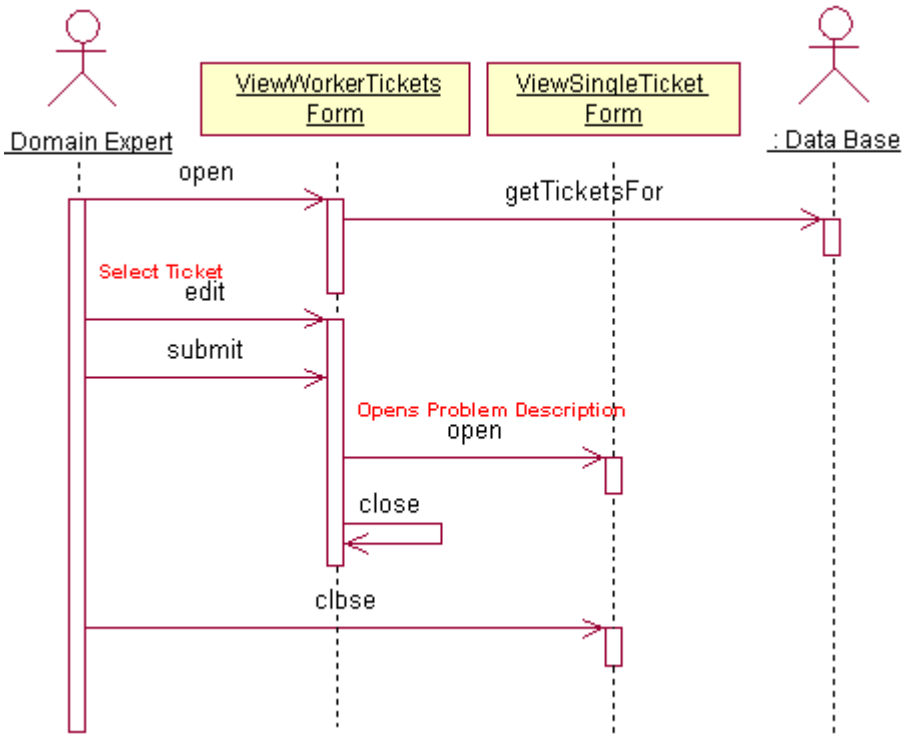


Fig. 2-9 Search tasks scenario.

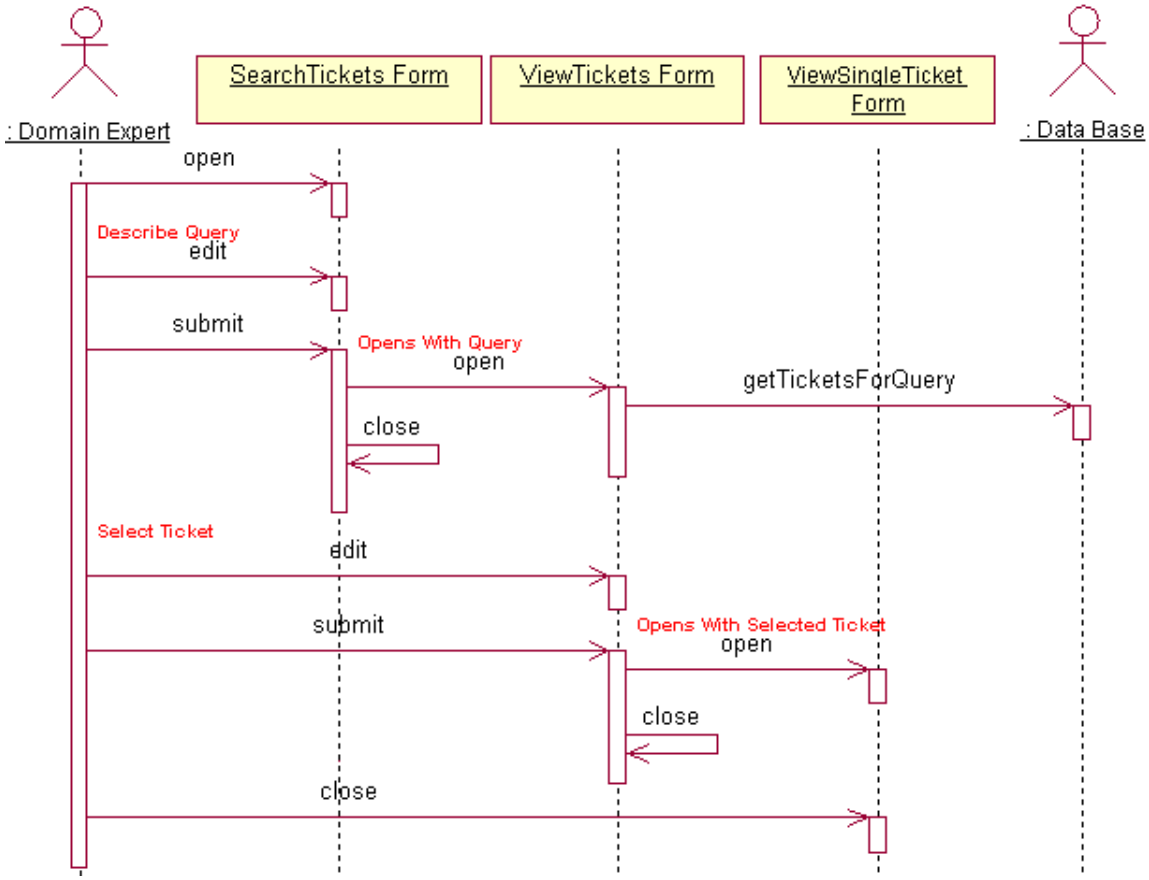


Fig. 2-10 Search tasks scenario.

Search Tasks use case starts when domain expert opens one of the two forms. Where the first form enables to make an advanced search among tickets see Fig. 2-10, and second shows all open tickets for a logged in expert see Fig. 2-9. Use case ends when the expert chooses one process ticket to work with.

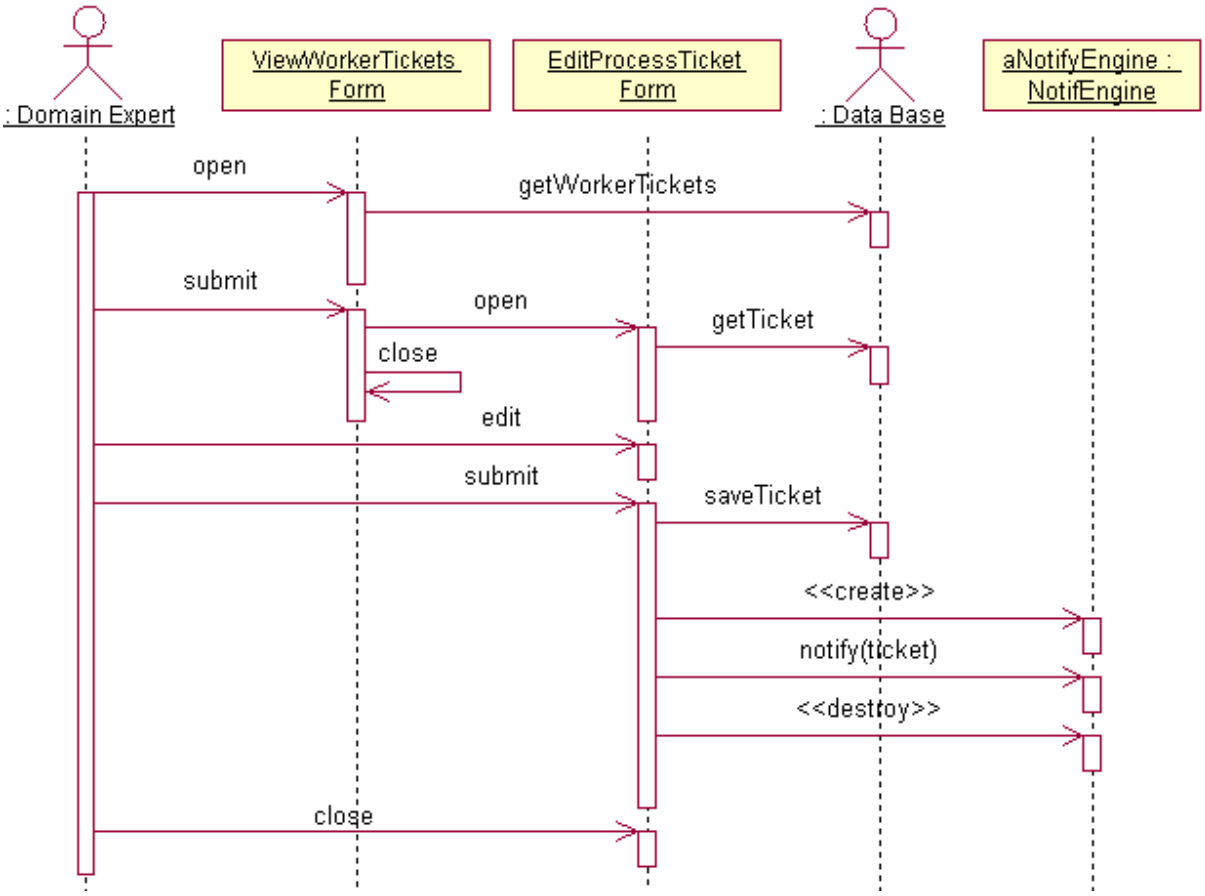


Fig. 2-11 Update ticket scenario.

Update Ticket use case starts when a domain expert opens a form to modify a process ticket or an activity. It ends when the expert submits changes. One possible flow of events for this use case is sown in Fig. 2-11.

3 System architecture

3.1 Introduction to software architecture:

Software System Architecture is a model of a software system that provides:

- A description of the high-level structure of the system, including the composition of the system's data and processing components, and the relationships and interconnections among the components.
- Guidance or rules on how new components should be added to the architecture, and how the architecture should be evolved over time.
- Rationale for why systems built using the architecture would satisfy stakeholder requirements.
- References to any standards or methods that are tied to or assumed by the architecture, whether mandated by stakeholders or chosen as foundation principles.
- A description of the high-level dynamic behaviour of the system, showing how the components would work together and synchronize their work over time to satisfy end-user scenarios. This would be especially important for large complex systems where behavioural issues (high-level state transitions, performance characteristics, etc.) were critical for understanding and guiding the system's construction.
- A refinement of the structure and behaviour showing allocations and relationships to physical hardware (processors, networking, etc.). This would be especially important if a common platform strategy is desired.

A component is any software system or subsystem that can be factored out and has a potentially standardizable or reusable exposed interface. Components in software architecture can be identified at different levels of abstraction, and the components identified at these different levels may not be in one-to-one correspondence. For example, viewing architecture at one level of abstraction, object services may be identified as components. Viewing the same architecture at a more detailed level, a given service may be implemented by several distinct software modules, which may be individually identified as components.

A good example of a low-level software component could be a container class *Vector* from JAVA libraries; Vector’s interface is just a set of methods. Example from a different level of abstraction could be the whole database system; the interface could be then defined as a special language, for example SQL.

Three-Tier architecture is a special type of client/server architecture consisting of three well-defined and separate processes, each running on a different platform:

1. The user interface, which runs on the user's computer (the client).
2. The functional modules that actually process data. This middle tier runs on a server and is often called the application server.
3. A database management system (DBMS) that stores the data required by the middle tier. This tier runs on a second server called the database server.

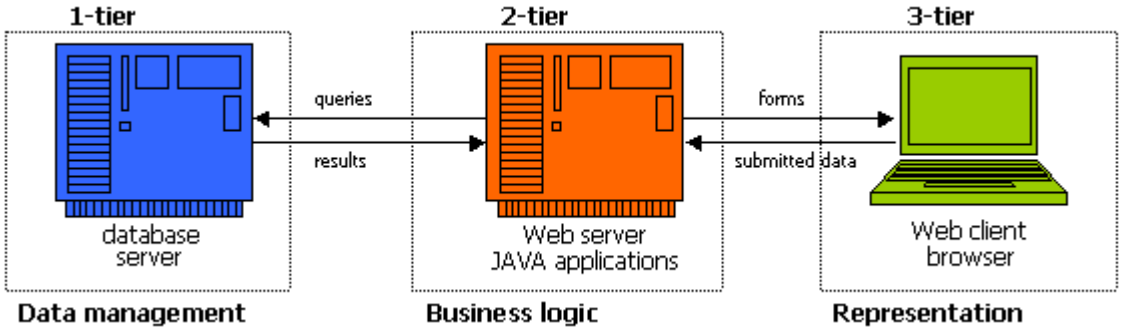


Fig. 3-1 Sample 3-tier system architecture.

The three-tier design has many advantages over traditional two-tier or single-tier designs, the chief ones is that the added modularity makes it easier to modify or replace one tier without affecting the other tiers. Separating the application functions from the database functions makes it easier to implement load balancing.

In Fig. 3-1 one of many possible 3-tier architecture is shown. All data are stored in the central database system. JAVA applications are running on a different server, and this is the business logic. Then the client connects to the web server, and the browser on his machine is responsible for forming data into something that the end user is able to recognise. If for example a different RBDMS is needed it can be easily replaced without changing the second and third tier.

A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It is a reusable design for all or part of a software system; a user interface framework only provides a design for the user interface of a system. By definition, a framework is an object-oriented design. It doesn't have to be implemented in an object-oriented language, though it usually is. Large-scale reuse of object-oriented libraries requires frameworks. The framework provides a context for the components in the library to be reused.

JAVA Applet is just one of many existing frameworks. An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application. The Applet class must be the super class of any applet that is to be embedded in a Web page or viewed by the Java Applet Viewer. The Applet class provides a standard interface between applets and their environment.

3.2 Design decisions

Rapid changes in business and market strategy can require a complete application or site overhaul as often as once a week, often forcing the developers to spend days changing hundreds of HTML pages. The difficulty of managing consistency across these pages has required a huge amount of time. Even if this less than ideal situation were acceptable, no computer developer wants to spend his or her life-making HTML changes to the pages.

With the advent of server-side Java, this problem has only grown. Servlet developers where forced to spend long hours modifying their Java code to produce required output.

The entire Java Server Pages (JSP) specification arguably stemmed from this situation. However, JSP is not a solution, as it only shifts the frustration to the HTML developer. In addition, JSP does not provide the clean separation between content and presentation it promises.

The main problem with standard web publishing is that in HTML pages content is mixed up with its representation. This implies that when the data change, also the representation has to be changed, not only the data itself. Also when the representation is modified, the data are edited too. I addition business logic has to be added to it somehow, and sometimes it is inserted within comments. Comments are not the right place for such an important system element.

The cure from this situation is to separate: data, business logic and representation into different layers. So changes in one layer do not affect other layers. To that purpose the whole set of XML and related technologies is created by the World Wide Web Consortium.

What was called for was a means to generate pure data content, and have that content uniformly styled either at predetermined times (static content generation), or dynamically at runtime (dynamic content generation). It is possible then to create within a company two different groups of employees, where first group will be responsible for creating content and second will be busy with representing that content. This two groups work in an independent manner, the representation group does not have to know that there are new data added to the system, also the content management group does not have to know how its data are represented.

The problem is that an engine must exist to handle content generation, particularly in the dynamic sense. Having hundreds of XML documents on a site does no good if there is no mechanism to apply transformations on them when requested. There are also business logic components, which have to affect this transformation.

A web-publishing framework attempts to address these complicated issues. Just as a web server is responsible for responding to a URL request for a file, a web-publishing framework is responsible for responding to a similar request; however, instead of responding with a file, it of then will respond with a published version of a file. In this case, a published file refers to a file that may have been transformed with XSLT, or converted into another format such as a PDF. The requestor does not see the raw data that may underlie the published result.

The whole work in an application is to perform business logic and then the XSL transformation to a XML document. So the developer has to decide, is he going to implement his own platform to perform transformation and logic to different XML

documents (lots of work and probably poor result in a short time) or he will search for an existing solution. The second approach has some advantages: the developer can consider his application far from XSLT details, the final product will be easier to maintain by others because the framework will be well documented and some developers will have an experience with it, finally there will be lots of ready to use supporting libraries and examples. This results in a decision to use one of the existing products.
The framework becomes the core TTS component. Web publishing using XML and XSL transformations is now a new technology, therefore there are not very many systems on the market and the list of good and stable ones is even smaller. Although Java language offers an easy interface into the various XML tools used by web publishing frameworks, additionally Java servlets offer a simple means of handling web requests and responses.
The most important feature of a system is its stability. Nobody wants even a great program, which is very fast and has many features, but breaks down once a day. Unfortunately the great majority of frameworks are version 1.0 or 1.x. While a higher version number is not a guarantee of stability, it often reflects the amount of time, effort, and review that a framework has undergone. As far as there is a decision of using Java as an implementation language there is no sense of investing time and money into platform-specific technologies. If the framework is tied to a platform (such as Windows), it is not a pure Java solution. The publishing framework must serve clients on any platform; why be content with a product that can't also run on any platform?
The last but not the least important factor is production presence of a framework. So it is important to determine if it is used in production applications. If it is the case that a vendor is not able to give a list of at least a few references, there is a risk of being a pioneer with a product.

The choice is to use an open source framework called COCOON from Apache Software Foundation, because of the following features:

- It is pure Java solution and therefore it is platform independent. And becomes a standard in open source world.
- COCOON is part of the Apache XML project and has default support for its technologies, which are de facto standards.
- It is now proven in many production installations.
- COCOON has support for World Wide Web Consortium new standard technologies; distinguished form some commercial solutions, which are working with company specific syntax in fact imitating W3C solutions.
- There is good support on a mailing list.
- It is public domain, so there are not any licensing costs.

3.3 TTS architecture

3.3.1 Overview

In Fig. 3-2 TTS architecture is shown. There are following conclusions from the diagram. There is defined a data model, which tells what exactly is a process ticket, an activity or what are the remaining data stored in the system. Data model is in package called Tickets. The data model of course affects RDBMS package, because it is the place where the model is directly implemented, but the shape of data affects also transactions package, because in TTS each transaction must know how does it's data looks like. There is a set of transactions defined in the system. Each transaction is responsible for performing business logic operations in certain scope. Depending on decisions taken in this package a data set is prepared. Transactions communicate with forms and form builder package. In fact transactions are executed in form builder, and there they are preparing data, based on these data and description of a form from forms package the resulting html code is generated, and then send to the user's browser. Forms package consists only of stylesheets. There is one stylesheet for one transaction on purpose to visualize it. The form builder is Cocoon itself.

The production system consists of several external components. They are Apache as a web server, Jakarta Tomcat as a servlet container, Cocoon as a web-publishing framework, Oracle as a relational database management system and of course a web client.

The web server is needed to serve others not dynamically generated elements, such as images, documentation or static content. In addition Apache is much more mature that Tomcat and integrating them both improves general system stability. Tomcat runs servlets inside Apache; there must be a servlet container because Cocoon is implemented simply as a servlet but particularly complex one.

n Fig. 3-3 one standard working cycle is shown. First when a browser sends a *request* Apache checks what is wanted. If it is possible just serve a file to satisfy browser it serves it, but in case when a dynamically generated content is asked Apache sends a *servlet request* to Tomcat. Then by analogy if Tomcat is asked to run a simple servlet it runs it and responses with generated data, but in case when a file with *XML* extension is requested Cocoon's service method is invoked.

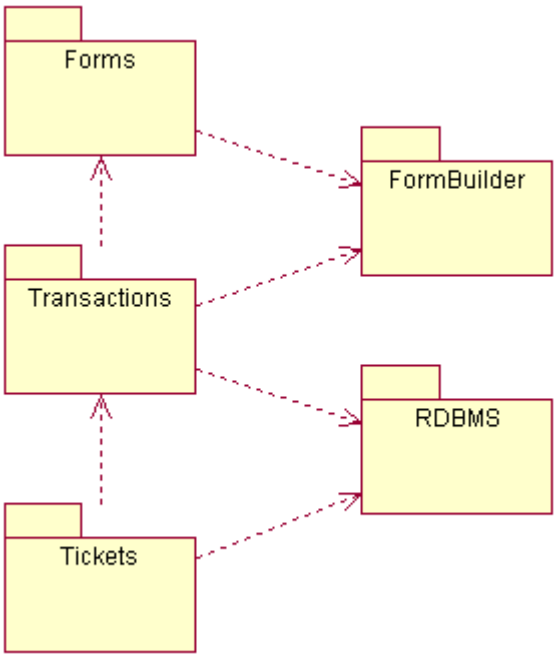


Fig. 3-2 TTS package diagram.

Inside Cocoon are special Java classes called Producers (*PageProcuders*) this classes encapsulates business logic. When there comes a request *populateDocument()* method of a producer is invoked. Inside this method an *XMLDocument* is created, this document will be then transformed with an appropriate stylesheet to produce an html output. Frequently to produce this document data from database are needed. To get them an *executeQuery()* method of an *SQLConnection* object is invoked, this method asks through JDBC bridge the database. Now, when the data are available, they can be added to the document. Usually there are also other elements for example current time, user name, some Ids and so on, added to the *XMLDocument*. When *populateDocument()* returns, *PageProducer* is destroyed and document is transformed with a stylesheet, which is read from a file. After transformation the document can be left, so the garbage collector will take care of it. The XSLT it the last stage of making a page, so the control is given back to Tomcat, which responds to Apache with the stream of generated data, and Apache sends this stream back to the browser.

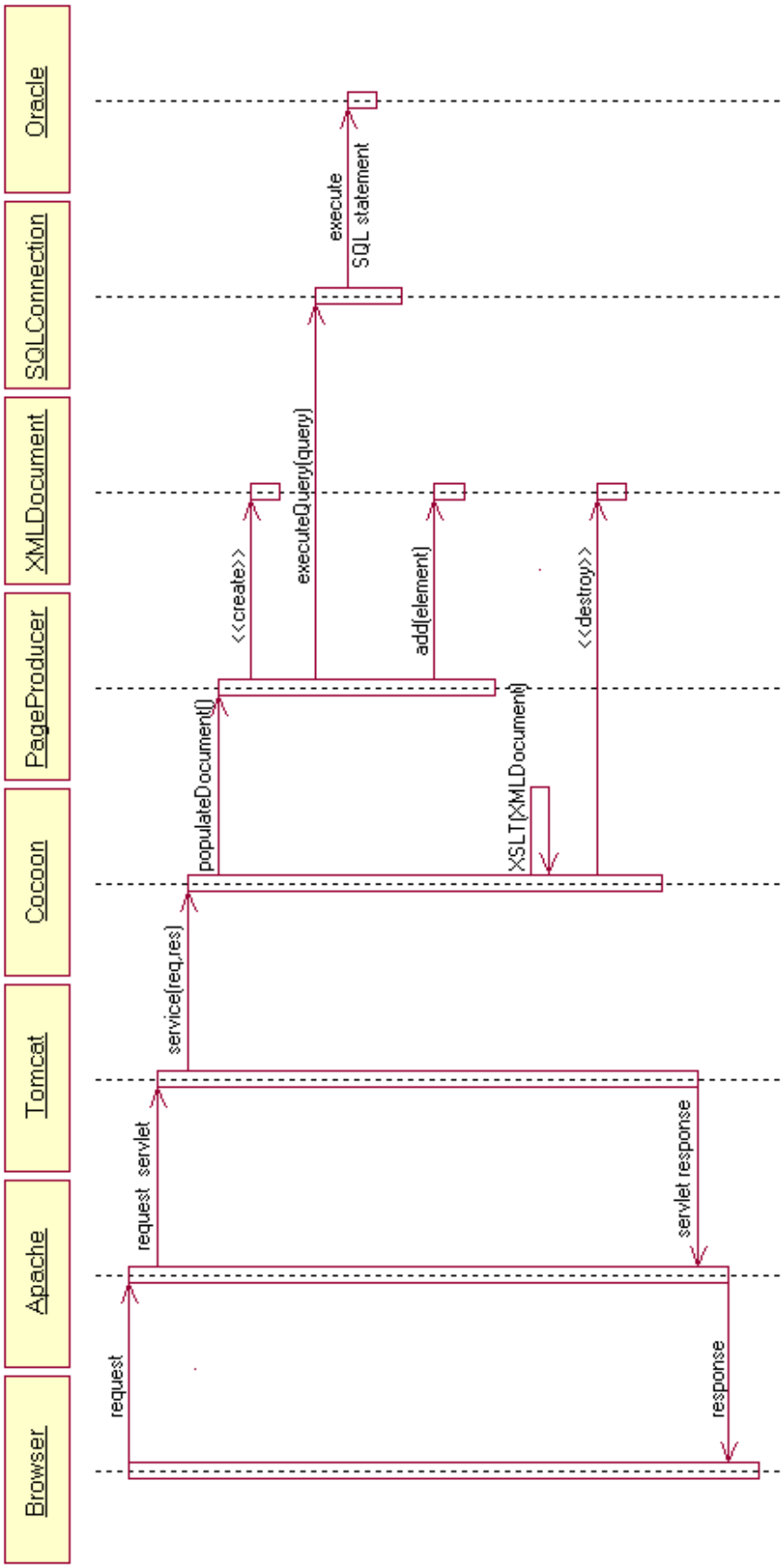


Fig. 3-3 High-level Cocoon operational diagram.

The couple of a transaction and its stylesheet are called a form. From the user point of view a form is just a browser with an open html form, which allows performing some well-defined task. For example: adding new ticket, adding a new activity to a ticket, closing tickets, deleting activities, logging into the system and so on. From the developers point of view a form is a pair of files. Where one file is an XSP page and second file is a stylesheet. There are some libraries, which are included into stylesheets or XSP pages, and are responsible for some specific tasks. An XSP library could be used for accessing connection pooling, ensuring security, building combo boxes and so on. A stylesheet library is rather a general template, which stores colours variables or templates used for formatting errors messages, tickets, activities and combo boxes in a standard way.

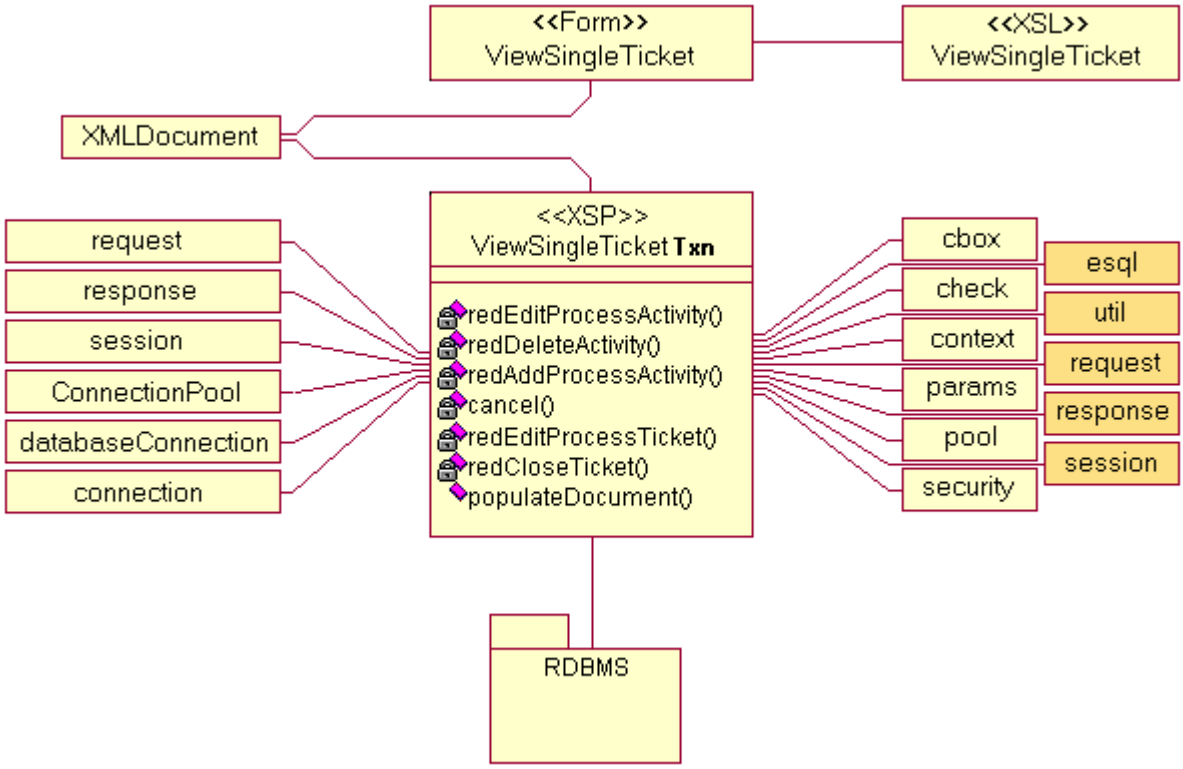


Fig. 3-4 High-level class diagram for *ViewSingleTicket* transaction.

In Fig. 3-4 is show a class diagram for *ViewSingleTicket* form. This diagram can vary a little bit, from one form to another, but the general idea is preserved. As it was told before there are two main elements, a stylesheet with stereotype *XSL*, and a transaction with stereotype *XSP*. The connection between these two independent parts is the *XMLDocument*, which is generated within transaction, and used together with stylesheet to produce html output. Transaction relies on information stored in a database, which is shown as a *RDBMS* package. Each transaction has access to following objects: *request* (HttpServletRequest), *response* (HttpServletResponse), *session* (HttpSession) which are passed from Tomcat to Cocoon and are parts of Servlet API

specification. Transactions, which are playing with database, have additional objects: *ConnectionPool*, *databaseConnection*, *connection*. These objects are used for communicating with database through connection pool, and are inserted into a transaction class thanks to *pool* library. There is a set of standard libraries: *esql*, *util*, *request*, *response*, *session* which are provided with Cocoon distribution. There are some others libraries written to be used in TTS: *cbox*, *check*, *context*, *params*, *pool*, *security*, these libraries are not tied to be used only with TTS; they are implemented to do some general tasks. For details take a look in Technical appendix.

3.3.2 Database model

TTS data model consists of sixteen tables. TTS is able to store information about any well-defined process. Process tickets are stored in HELPDESK_PROCESSTICKETS table. To each process ticket could be assigned any quantity of activities. Activities are stored in HELPDESK_PROCESSACTIVITIES table. Both process tickets and activities have assigned a worker who is responsible of it. Workers data are stored in HELPDESK_WORKERS table. A process ticket is just a representation of a problem, which happened in a real world, so there is a client who has it, information about clients, is stored in HELPDESK_CLIENTS. From organizational reasons clients are divided into groups, these groups are stored in HELPDESK_CLIENTTYPES. With process tickets is connected information if its client wants the TTS to send an email in case of closing or adding new activities to a ticket HELPDESK_NOTIFTYPES table. There are also keywords to make an easy associations HELPDESK_KEYWORDS table. Is also a list of computers in DESY HELPDESK_COMPUTERS table. And a list of all supported programs HELPDESK_PROGRAMS table.

TTS allows storing information about different types of process. There is a table called HELPDESK_PROCESSTYPES where a type of a process is defined. What makes differences between processes it their name, their set of allowed statuses and their set of allowed activities. Readable status names are stored in HELPDESK_STATUS table; it is made on purpose to reduce redundancy among statuses. When a new process type is added, there must are new entries in HELPDESK_PROCESSSTATUS to allow this process type to have an allowed set of statuses (open, closed, paused, etc.). The same is made when adding new activities. Each activity has its type. Types of activities are stored in HELPDESK_ACTIVITYTYPES table. But there is connection between activity type and process type. One activity type could be instantiated only belonging to one type of process. Allowed statuses for a certain type of activity are stored in HELPDESK_ACTIVITYSTATUS table.

In TTS data model are stored also information about user roles. There is a table called HELPDESK_ROLES where each entry defines a new role. The name of a role is used then in XSP transactions. Each transaction defines a set of roles. Workers who possess at least one role from the list could perform this transaction. Connection between roles and workers is build up in HELPDESK_WORKERROLES table, where are stored workers and roles ids.

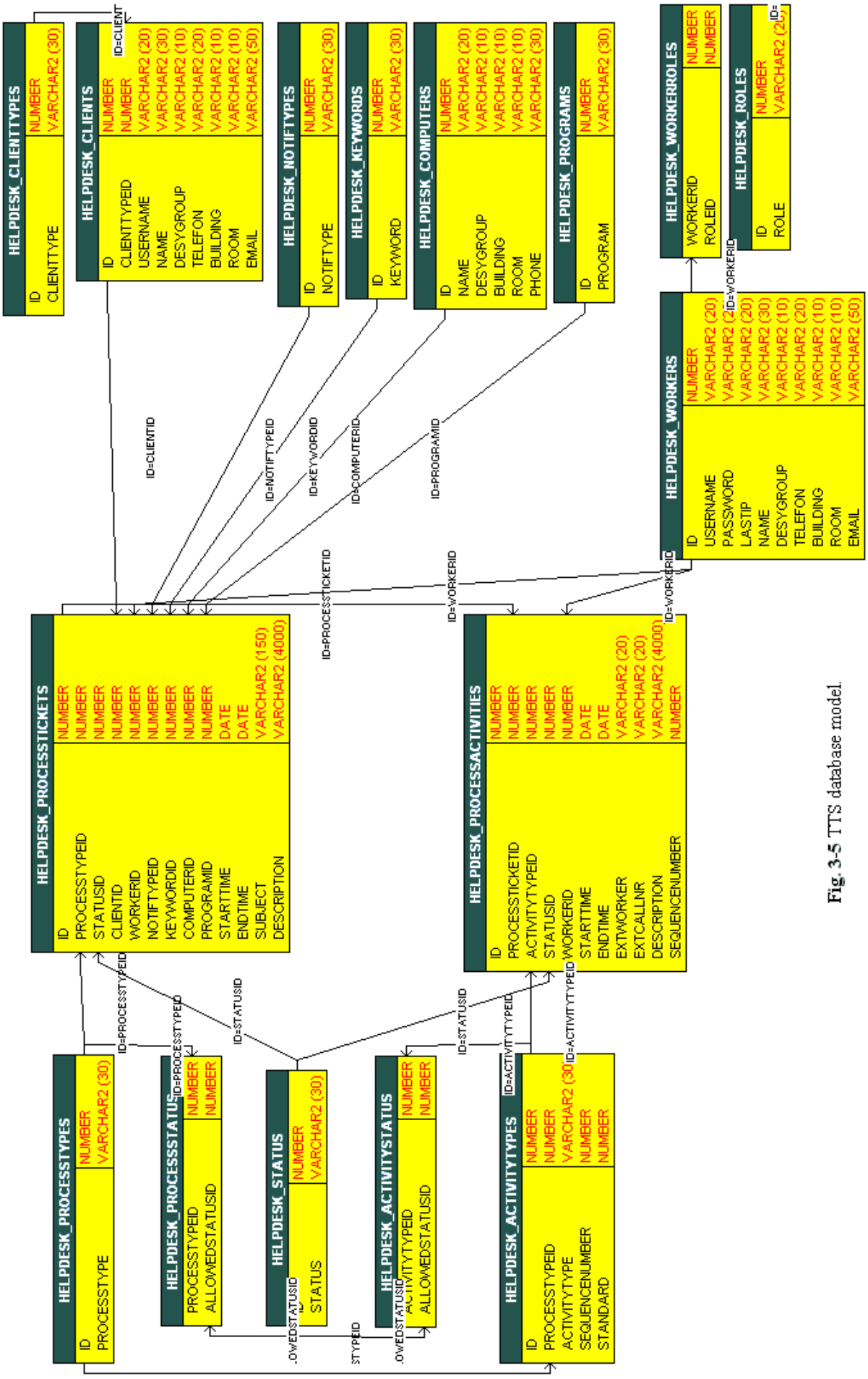


Fig. 3-5 TTS database model

Rules for updating the database:

If there are already process tickets of certain type in the database, and the process definition needs to be changed. Changing allowed activities or activity types will affect existing process tickets. To avoid changes to existing data, mark activities as obsolete and add new ones. The same applies to allowed statuses, if they must be preserved, mark them as obsolete and add new ones. The same rule applies to the rest of the configurations tables.

3.3.3 COCOON components

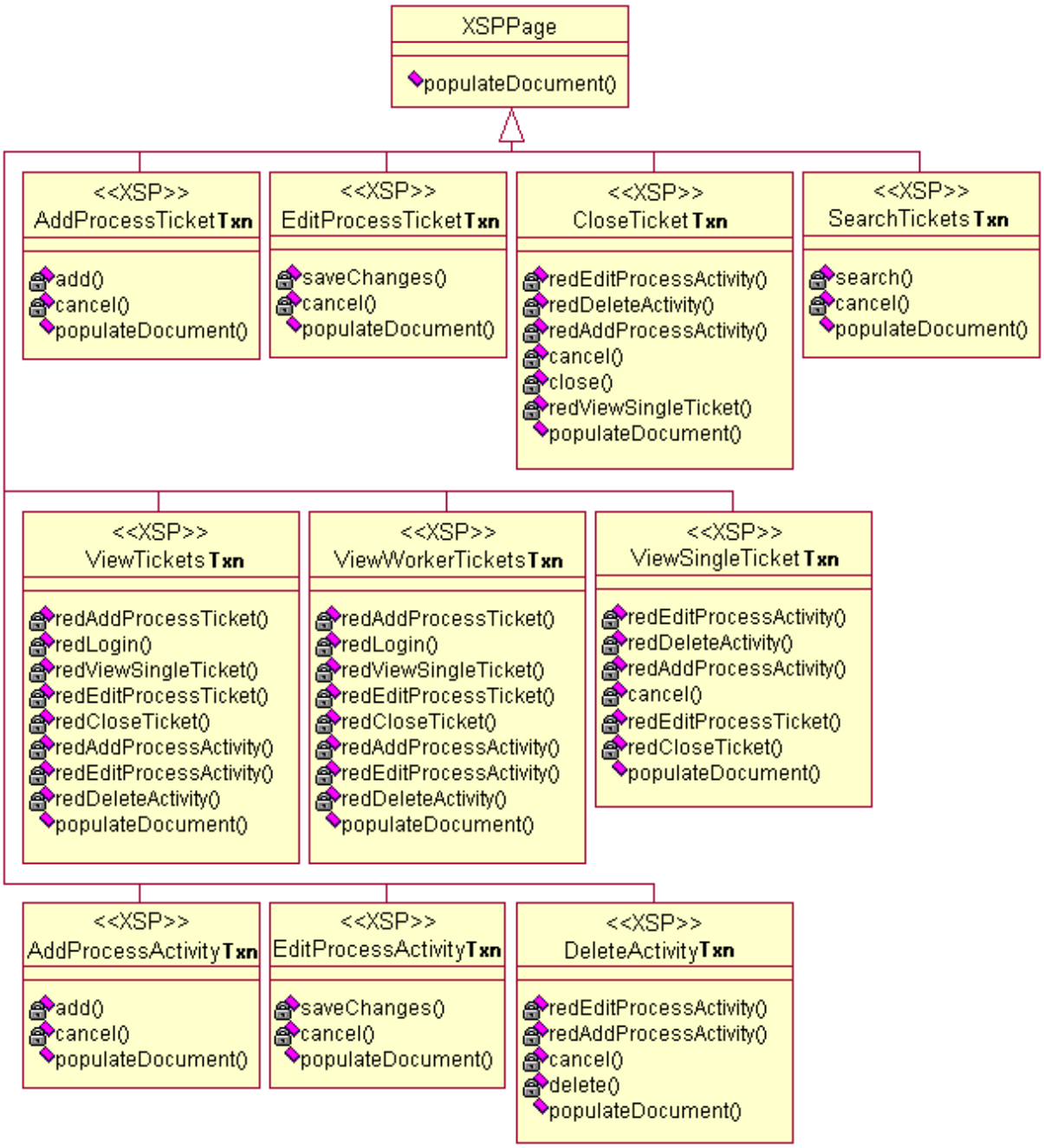


Fig. 3-6 Cocoon transactions diagram.

The developer sees only two types of components: XSP pages and stylesheets. Fig. 3-6 shows transactions diagram. In terms of Cocoon's terminology XSP pages are called sometimes logic sheets.

List of all forms (XSP page and stylesheet):

XSP page	Stylesheet	Description
AddProcessTicket.xml	AddProcessTicket.xsl	Adding new process ticket.
EditProcessTicket.xml	EditProcessTicket.xsl	Editing process ticket.
CloseTicket.xml	CloseTicket.xsl	Closing process ticket.
SearchTickets.xml	SearchTickets.xsl	Searching tickets.
ViewTickets.xml	ViewTickets.xsl	Viewing a list of tickets.
ViewWorkerTickets.xml	ViewWorkerTickets.xsl	List of responsibilities.
ViewSingleTicket.xml	ViewSingleTicket.xsl	Ticket and its activities.
AddProcessActivity.xml	AddProcessActivity.xsl	New activity to a process.
EditProcessActivity.xml	EditProcessActivity.xsl	Editing existing activity.
DeleteActivity.xml	DeleteActivity.xsl	Removing activity.
Login.xml	Login.xsl	Loging into system.

List of all general stylesheets:

cbox.xsl	Transforms data into a combo box.
error.xsl	Transforms error description into an error message.
style.xsl	Stores colours data.
tickets.xsl	Transforms a general process ticket description.

List of all XSP libraries:

cbox.xsp.xsl	Prepares and caches data for combo.xsl.
check.xsp.xsl	Checks input parameters against fulfilling a condition.
context.xsp.xsl	Provides support for introducing limited context to http.
params.xsp.xsl	Supports passing on input parameters to outputed html.
pool.xsp.xsl	Gives access to database connections using pooling.
security.xsp.xsl	Allows or denides access depending on user roles.
login.xsp.xsl	Supports login process.

3.3.4 Multi language support discussion

There is a requirement from the supervisor of making the TTS multilingual. Unfortunately it appeared that it is not easy to do so. There are two architecture proposals of system with multi language support and one without it.

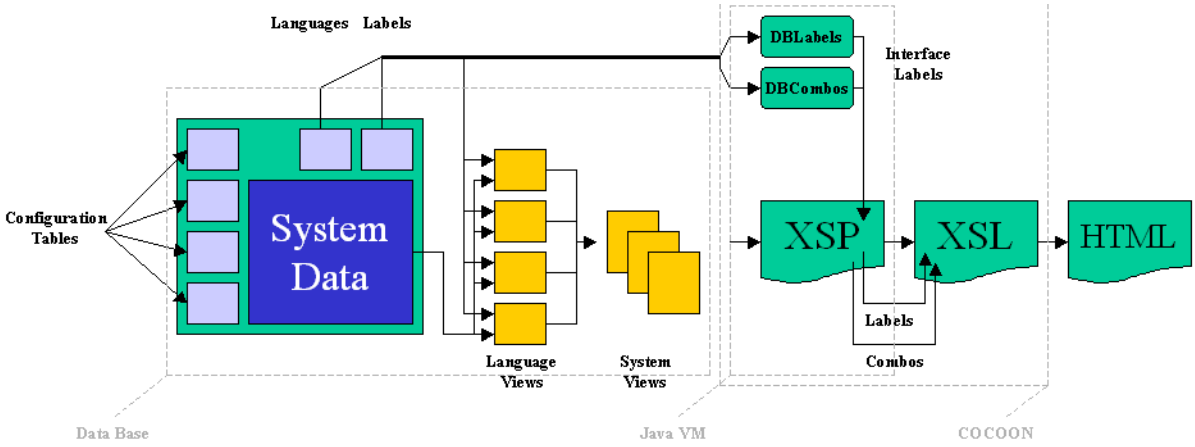


Fig. 3-7 First system variant.

In Fig. 3-7 is the first proposal. It includes changing the data model, so there is no open text it the configuration tables, instead of it there are two more tables introduced *(Languages Labels)*, one of them includes information about which languages are supported and what are their names, in second are stored all text labels present in the system. To give an easy way to get the data out of this model a few database views are made *(Language Views)* which are sitting on top of configuration tables, one view per one language dependent table. There is second generation of views, this views depend on the first ones, and gives the whole process tickets or activities description directly to the business logic layer. Labels in interface also have to be translated. In business logic language dependent data form cache classes called *DBLabels* (text labels) and *DBCombos* (data for combo boxes) are added to the resulting *XMLDocument*. When the *XMLDocument* is translated to get the html, all labels ids from stylesheets are replaced with their representation in active language.

Advantages:

1. All language dependent data are stored in one place, which is a database table.
2. There is provided an easy to use tool for adding new language support.
3. It is fast, because there are two memory caches for labels.

Drawbacks:

1. Administrator has to use a special tool for adding and modifying labels. Working directly with a table is difficult because of the data format. So he has to learn it first.
2. The configuration table's data consists of ids, which are numbers; there are no more text descriptions, which can guide somebody who is modifying them. When there is a need to reconfigure system, changes must be done also to labels table, which is tricky and could cause difficult errors.
3. There are several views in the database, which are building to perform join queries between configurations tables and labels table. In addition there is Cartesian product calculated which is slowing down performance.

4. Interface labels are included in the *XMLDocument* what slows performance.
5. In the XSL stylesheet there are no textual labels, instead of them, there are templates, which replaces itself with appropriate label. This makes the whole stylesheet more difficult to read and maintain, especially for non-developers.
6. The language transformation code is scattered in all places in the system. Instead of it, there should be one language transformation layer.
7. After modifying system configuration the label and combo cache must be invalidated.

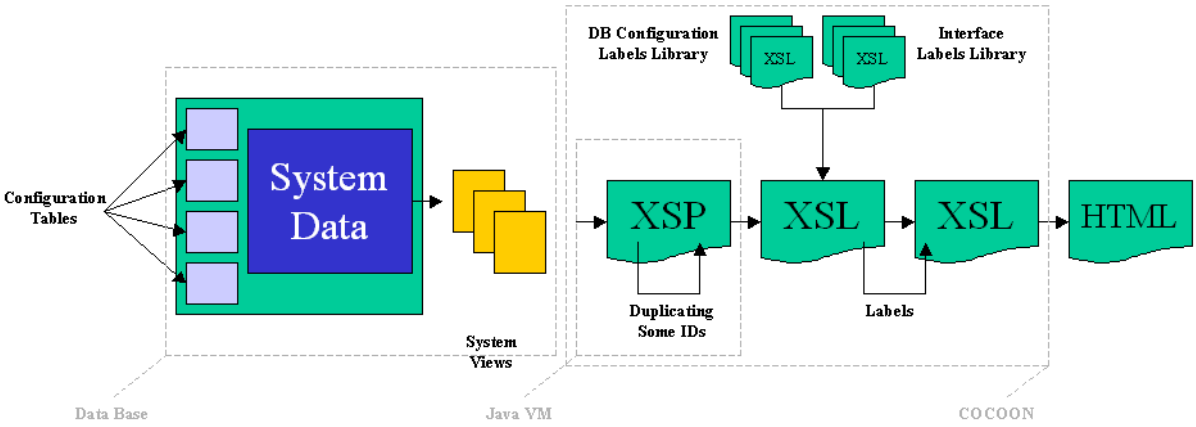


Fig. 3-8 Second system variant.

In Fig. 3-8 is the second proposal. In this variant the language dependent information is moved from database into special stylesheets library. Data taken out of the tables do not have textual information, but ids. These ids are then copied in the business logic layer to the XMLDocument. There is one more transformation added, where all ids in the source document are replaced with theirs language dependent representation. This solution introduces a pure language transformation layer.

Advantages:

1. There is a special language transformation layer in the system, which does not affects its others parts.
2. All labels are stored in XSL files, which are in pure text format.
3. Queries results are language independent.
4. There is no special cache mechanism.

Drawbacks:

1. Configuration changes must be made in two places parallel, in database and in appropriate stylesheet. This causes synchronization difficulty, and may sometimes results in heavy errors.
2. The *XMLDocument* is not build in the intuitive way. It is because of some indexes, which must be repeated to be replaced with text in language translation layer.
3. There are two consecutive XSL transformations that could cause great performance reduction.

4. How in the first variant, the interface labels, all have to be inserted in *XMLDocument*. The html forming stylesheet again does not have open text inside.

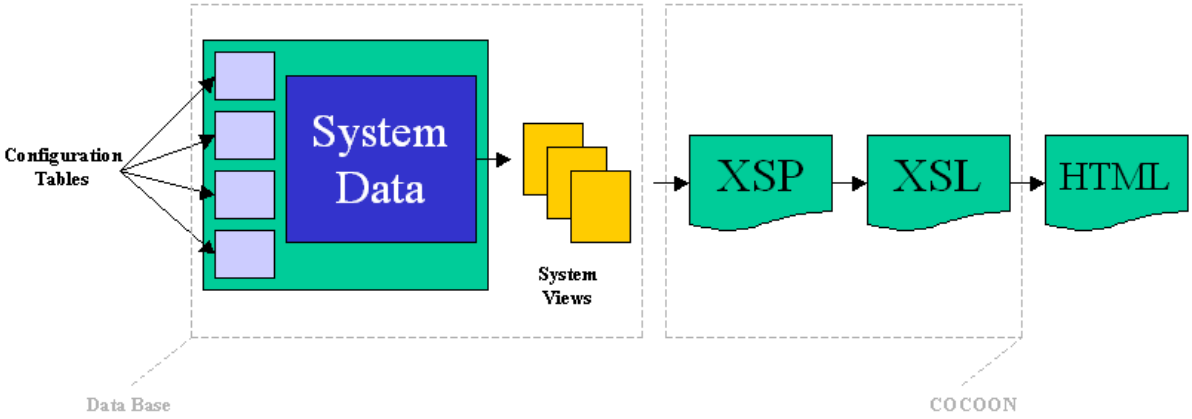


Fig. 3-9 Third system variant.

In Fig. 3-9 is the third proposal. This variant does not support multi languages.

Advantages:

1. Configuration tables will contain labels, so making an adaptation becomes a doable task. Referential integrity will guard data from errors.
2. There is only one XSL transformation, which will perform well.
3. The stylesheets becomes simpler. They will also contain text labels, which will guide administrators by making changes.
4. The whole system becomes much simpler, is very important by first usage of a new technology. The maintenance work will be much easier. It is the fastest version.

Drawbacks:

1. Multi language is not supported.

The requirement of multi language support is of low priority, and it has very high risk. So there is a decision of dropping it from the implementation.

3.4 External TTS components

Web publishing framework COCOON

Cocoon is a 100% pure Java publishing framework that relies on new W3C technologies (such as DOM, XML, and XSL) to provide web content. The Cocoon project aims to change the way web information is created, rendered and served. The new Cocoon paradigm is based on the fact that document content, style and logic are often created by different individuals or working groups. Cocoon aims for a complete separation of the three layers, allowing them to be independently designed, created and managed, reducing management overhead, increasing work reuse and reducing time to market.

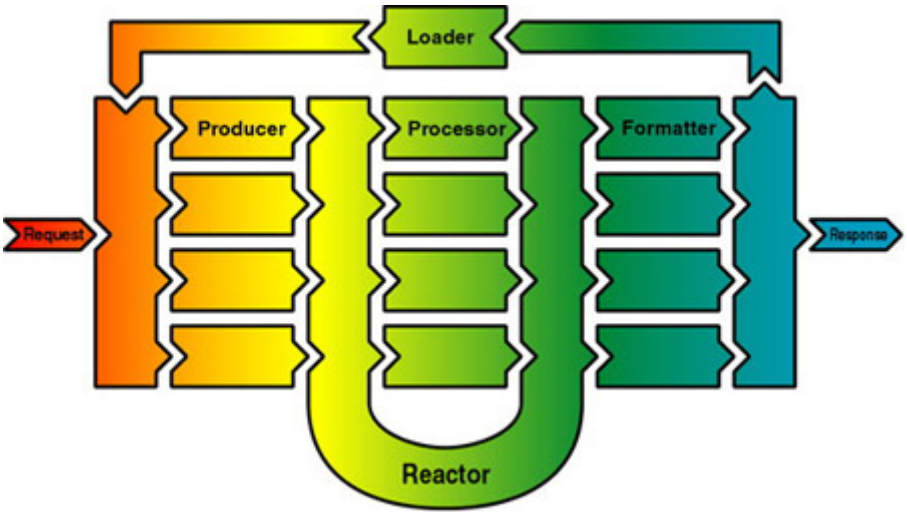


Fig. 3-10 COCOON architecture.

Description of components in Fig. 3-10:

- **Request** – Wraps around the client's request and contains all the information needed by the processing engine. The request must indicate which client generated the request, which URI is being requested and which producer should handle the request.
- **Producer** – Handles the requested URI and produces an XML document. Since producers are pluggable, they work like subservlets for this framework, allowing users to define and implement their own producers. A producer is responsible is responsible for certain the XML document which is fed into the producing reactor. It is up to the producer implementation to define the function that produces the document from the request object.
- **Reactor** – Responsible for evaluating which processor should work on the document by reacting on XML processing instructions. The reactor pattern is different from a processing pipeline since it allows the processing path to be dynamically configurable and it increases performance since only those required processors are called to handle the document. The reactor is also responsible for forwarding the document to the appropriate formatter.
- **Formatter** – Transforms the memory representation of the XML document into a stream that may be interpreted by the requesting client. Depending on other processing instructions, the document leaves the reactor and gets formatted for its consumer. The output MIME type of the generated document depends on the formatter implementation.
- **Response** – Encapsulates the formatted document along with its properties, such as length, MIME type, etc.
- **Loader** – Responsible for loading the formatted document when this is executable code. This part is used for complied server pages (principally XSP) where the separation of content and login is merged and compiled into a Producer. When the formatter output is executable code, it is not sent back to the client directly, but gets loaded and executed as a document

producer. This guarantees both performance improvement (since the producers are cached) as well as easier producer development, following the common compiled server pages model.

Processing instructions are:

<?cocoon-process	type="xxx"?>	for processing, and
<?cocoon-format	type="yyy"?>	for formatting

In a complex server environment like Cocoon, performance and memory usage are critical issues. Moreover, the processing requirement for both XML parsing, XSLT transformations, document processing and formatting are too heavy even for the lightest serving environment based on the fastest virtual machine. For this reason, a special cache system was designed underneath the Cocoon engine and it is able to cache both static and dynamically created pages.

Cache operation is simple but rather powerful:

- when the request comes, the cache is searched
    - if the request is found:
        - its changeable points are evaluated
        - if all changeable points are unchanged
            - the page is served directly from the cache
        - if a single point has changed and requires reprocessing
            - the page is invalidated and continues as if it was not found
        - if the request is not found
            - the page is normally processed
            - it is sent to the client
            - it is stored into the cache

The special cache system is required since the page is processed with the help of many components, which, independently, may change over time. For example, a stylesheet or a file template may be updated on disk. Every processing logic that may change its behaviour over time it is considered *changeable* and checked at request time for change.

Each changeable point is queried at request time and it is up to the implementation to provide a fast method to check if the stored page is still valid. This allows even dynamically generated pages (for example, an XML template page created by querying a database) to be changed and, assuming that request frequency is higher than the resource changes, it greatly reduces the total server load.

Moreover, the cache system includes a persistent object storage system, which is able to save stored objects in a persistent state that outlives the JVM execution. This is mainly used for pages that are very expensive to generate and last very long without changes, such as compiled server pages.

The store system is responsible for handling the cached pages as well as the preparsed XML documents. This is mostly used by XSLT processors, which store their stylesheets in a pre-parsed form to speed up execution in those cases where the original file has changed, but the stylesheet has not (which is a rather frequent case).

TOMCAT servlet container

Tomcat is the official Reference Implementation for the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed by Sun under the Java Community Process.

Tomcat is developed in an open and participatory environment and released under the Apache Software License. Tomcat is intended to be a collaboration of the best-of-breed developers from around the world.

Apache HTTP Server

The Apache is open-source HTTP server for various modern desktop and server operating systems, such as UNIX and Windows NT. Apache is a secure, efficient and extensible server, which provides HTTP services in synchronization with the current HTTP standards. Because Apache, Tomcat and Cocoon are just a family of products from the same open source foundation, they are well integrated.

RDBMS Oracle

Oracle in terms of TTS is just a relational database management system, which implements SQL. Oracle is very stable and powerful platform, but could be replaced with any other database with SQL and JDBC implementation.

Java Virtual Machine

Java virtual machine is a specification for software, which interprets Java programs that have been compiled into byte-codes. The JVM instruction set is stack-oriented, with variable instruction length. Unlike some other instruction sets, the JVM's supports object-oriented programming directly by including instructions for object method invocation. Because JVM is an implementation of an interpreter, it is possible to execute Java programs on every operating system, which has a version of virtual machine, designed to run in it. Thanks to it TTS can perform on every serious type of server.

4 Implementation

4.1 Implementation decisions

Each computer system has to be well integrated into the destination environment. It does not matter if it is a big airport or just one desktop PC connected to the Internet via modem. TTS is not an exception from this rule. Analysing DESY and IPP requirements for that reason results in following conclusions:

Performance:

The great majority of web-based applications in DESY uses common gateway interface (CGI) with Perl as a programming language. However this solution has some advantages, for example many workers know how to use Perl, there are some drawbacks. The main is that CGI is rather slow, especially on a slow web server. Low performance of an application could result in low users acceptance. Therefore there is a need to find another solution. Java offers an API called Servlet. Servlet is a Java class, which can be executed inside a servlet container, which is a part of every Java enabled web server. Servlets offer good performance. There are two reasons why Servlets are performant: first today's Java virtual machines can optimise executed code which can perform as fast as C++, second servlets eliminate waste of time which is needed for loading and executing an CGI application, because a servlet which is once loaded stays in the memory ready to serve other clients. As far as servlet can stay in memory it can handle open database connection, that eliminate the need of opening a new connection for each request.

Maintability:

Today nobody will be able to sell a system in which introducing changes will be extremely difficult. Or a system which configuration is hard coded in thousands lines of code. Some classical languages such as C or C++ offer great opportunities to write code, which is difficult to understand or to maintain. Presence of pointers allows programmers to write very tricky code, also pointers force programmers to be busy with memory allocation. There is a lack of garbage collector, which can assure that there will not be any memory leaks. Therefore there is a decision to use JAVA as an implementation

language. Java allows writing very readable code without pointers and memory allocation. Java also is an object oriented language from its beginning with great support for objectivity. Also presence of many standard libraries makes programming much easier. But there is a problem; DESY does not have JAVA developers. There is also a requirement, which tells that somebody who is not familiar with core TTS technology Java should be able to do maintenance or extending operations to the system. It seems to be an unworkable task, but the doors are open thanks to a new quality, which comes with XML.

There are also other problems, which need to be solved. The usual way of producing server side applications is to develop them on a test machine. After reaching sufficient reliability level it can be moved to a production server. It is fully possible that there will be a need to install TTS on different type of production servers. Once it could be Windows NT machine the second time Sun Solaris. Workstations usually are with Windows NT operating system. This problem is solved automatically with choosing JAVA language thanks to its main assumption of platform independence.

Availability:

DESY has about two thousands of workstations. There are many different operating systems such as: UNIX, Sun Solaris, Linux or Widows, working on many different platforms, for example: SGI, SPARC, INTEL. Nobody wants to develop a special version of an application, to use it on each different system type. It is possible to write a single Java application and install it on workstations. But it is not convenient, before application the virtual machine must be installed. There are systems, which need installing many patches required to run Java on them. Also it is not exactly known from which workstations users will access TTS. Fortunately there is one more solution. On each system a working web browser is installed. Nothing more is needed that to instantiate a single web server with Java inside, and this automatically makes an application ready to use on every computer in DESY. This is exactly what is wanted. So there is a decision to develop web-based application. But there are several restrictions. A few employees in DESY are blind and have a Braille monitors. Also a few users will use lynx for accessing TTS. There are also many browsers, which do not support JAVA or support an old version; therefore Java applets should not be used. As a result, there is one more conclusion to use only pure HTML solution.

Stability

Computer system once installed should require as little work as possible. There should are some safety mechanisms, which could guarantee that a system will operate. For example if there are errors, information that they happened should be written to a log file, and if system hangs up or will be closed it should be automatically restarted. Using Java greatly increases stability. Java virtual machine isolates running code from low-level hardware. So the possibility to damage web server or operating system is greatly decreased. Also it is possible

to install a security manager, which will protect host system from dangerous actions taken by badly written programs.
Using servlets also increases stability; it is because of servlets live cycle. When a fatal error occurs in a servlets an exception is throws. This exception goes up to the servlet container, which terminates the servlet. But when a new request will come to killed servlet it will be once again loaded into memory and will operate, unless this critical situation happens again.
Database operations are also possibly source of problems. Some native interfaces such as ODBC can even hang up the whole system. Java offers an API, which is called Java Database Connectivity (JDBC). JDBC could be used in two variants. First with usage of native methods. Second as a pure Java solution, which uses *JDBC Thin Driver*, which has all advantages of isolated Java code.

4.2 Used technologies

4.2.1 JAVA and JAVA Servlets

Java servlets are designed on purpose to replace CGI programs. They offer better performance and safety. In this subsection an example of a small servlet will be shown. Next to the example there will be a description of how servlets are integrated into the web server environment.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(
            HttpServletRequest request,
            HttpServletResponse response
    )
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<head>");
        out.println("<title>Hello World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Fig. 4-1 Source code for “Hello World” example servlet.

HttpServlet provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:

- doGet, if the servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests
- doDelete, for HTTP DELETE requests
- init and destroy, to manage resources that are held for the life of the servlet
- getServletInfo, which the servlet uses to provide information about itself

There is almost no reason to override the service method. Service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the doXXX methods listed above). Likewise, there's almost no reason to override the doOptions and doTrace methods.
Servlets typically run on multithreaded servers, so a servlet must handle concurrent requests and synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

To communicate with the web server environment, two types of objects are used: HttpServletRequest and HttpServletResponse. HttpServletRequest extends the ServletRequest interface to provide request information for HTTP servlets. HttpServletResponse extends the ServletResponse interface to provide HTTP-specific functionality in sending a response, for example, it has methods to access HTTP headers and cookies. The servlet container creates both an HttpServletRequest object and an HttpServletResponse object and passes them as arguments to the servlet's service methods (doGet, doPost, etc).

In Fig. 4-1 the simplest possible servlet is shown. The essential thing in working with http servlets is that it always goes the same way. First, HelloWorld servlet is a subclass of *HttpServlet*, and implements *doGet* method. Second, a PrintWriter is taken from response object. Then the generated html text is written into it. The result is shown in Fig. 4-2. It is visible, that introducing even a small change requires editing Java code and its recompilation.

```
<html>
<body>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

Fig. 4-2 Html code generated in “Hello World” example servlet.

What is worth mentioning about the *HttpServlet* is that it is a subclass of more generic class called *Servlet*. It is an implementation of a servlet used to work with HTTP. It is possible to write a servlet, which will be able to work with any other protocol.

4.2.2 XML and XSL

Development of XML started in 1996 and it is a W3C standard since February 1998. The designers of XML simply took the best parts of SGML, guided by the experience with HTML, and produced something that is no less powerful than SGML, but vastly more regular and simpler to use.

XML is interesting because of two features. It allows to easy representing structured data in a text format and it has a mechanism, which is provided to transform between different formats of XML or even between XML and other binary or text data formats.
An example of XML is in Fig. 4-3. It is standard internal format of a query result set in TTS. To visualize this data a stylesheet in Fig. 4-4 is provided. This stylesheet uses XSLT to produce an html document, which has a table formatting, where one row is one record from the database. The result interpreted in a browser is shown in Fig. 4-5.

```
<?xml version='1.0'?>

<?xml:stylesheet type="text/xsl" href="Library.xsl"?>

<ROWS>
<ROW ID="1">
      <NAME>w01ntcad3</NAME>
      <DESYGROUP>IPP</DESYGROUP>
      <BUILDING>2b</BUILDING>
      <ROOM>32</ROOM>
</ROW>
<ROW ID="2">
      <NAME>ipppub3</NAME>
      <DESYGROUP>IPP</DESYGROUP>
      <BUILDING>2b</BUILDING>
      <ROOM>32</ROOM>
</ROW>
</ROWS>
```

Fig. 4-3 Sample XML structure of data taken out from HELPDESK_COMPUTERS table.

It is important to say that when new rows will be added to the source data, the stylesheet will transform them. In case when a new format will be needed, not a table but for example a combo box, the data stays intact, there are changes only in the stylesheet.

```
<?xml version='1.0'?>
```

```
<xsl:stylesheet
      xmlns:xsl="http://www.w3.org/TR/WD-xsl"
      xmlns="http://www.w3.org/TR/REC-html40"
      result-ns="">

<xsl:template match="/">
      <table border="1" bgcolor="#FFFFCC">
          <tr>
          <th>name</th>
          <th>group</th>
          <th>building</th>
          <th>room</th>
          </tr>
          <xsl:apply-templates select="/ROWS/ROW"/>
      </table>
</xsl:template>

<xsl:template match="ROW">
      <tr>
      <td><xsl:value-of select="NAME"/></td>
      <td><xsl:value-of select="DESYGROUP"/></td>
      <td><xsl:value-of select="BUILDING"/></td>
      <td><xsl:value-of select="ROOM"/></td>
      </tr>
</xsl:template>

</xsl:stylesheet>
```

Fig. 4-4 Stylesheet designed to represent data in a table.

name	group	building	room
w01ntcad3	IPP	2b	32
ipppub3	IPP	2b	32

Fig. 4-5 Result of the transformation viewed in an html browser.

The transformation pattern is shown in Fig. 4-6. When an XML transformation is needed, both documents the XML and XSL are read into memory. When the document is read it is formed in a tree. Then both trees are processed in XSLT Engine called sometimes an XSL processor. The result of a transformation is another tree, which can be then used inside an application or send to a file or over network.

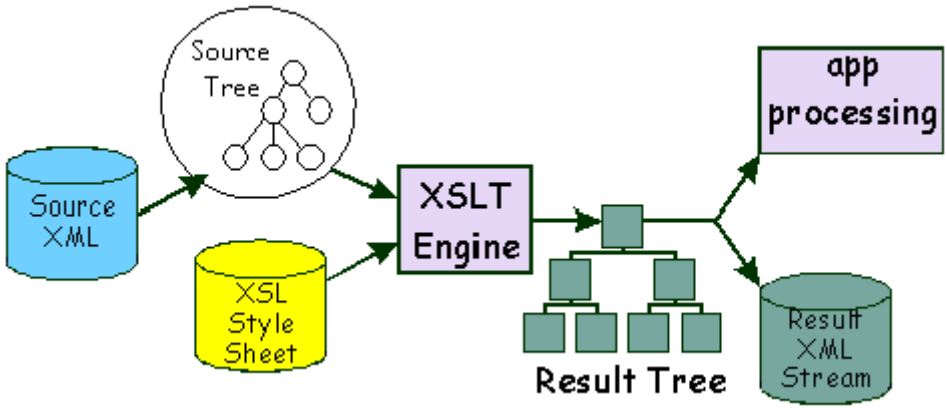


Fig. 4-6 XSL transformation pattern.

4.2.3 XSP

XSP (eXtensible Server Pages) is Cocoon's technology for building web applications based on dynamic XML content. Beyond static content, web applications demand dynamic content generation capabilities, where XML documents or fragments are programmatically produced at request time. In this context, content is the result of computations based on request parameters and, often, on access to external data sources such as databases or remote server processes. This distinction in content origin extends the "traditional" regions of web publishing (content and presentation) to also encompass that of logic. Dynamic web content generation has traditionally been addressed by embedding procedural code into otherwise static markup. This approach is fully supported by XSP. An XSP page is a Cocoon XML document containing tag-based directives that specify how to generate dynamic content at request time. Upon Cocoon processing, these directives are replaced by generated content so that the resulting augmented XML document can be subject to further processing (typically an XSLT transformation). XSP pages are transformed into Cocoon producers, typically as Java classes, though any scripting language for which a Java-based processor exists could also be used. Directives can be either XSP built-in processing tags or user-defined library tags. XSP built-in tags are used to embed procedural logic, substitute expressions and dynamically build XML nodes. User-defined library tags act as templates that dictate how program code is generated from information encoded in each dynamic tag.

In Fig. 4-7 and Fig. 4-8 are examples of two identical XSP pages. This pages shows a caption informing how many times the page was accessed. This is done very simply. As was told before; an XSP page is converted into a Java class called a producer. Inside this class a static variable is declared and initialized:

```
static private int counter = 0;
```

Then a simple access method is provided, which returns actual value of the counter and increases it:

```
        private synchronized int count() {
        return counter++;
    }
```

It must be synchronized because it is possible that it can be accessed concurrently by a few parallel requests. In the right place in the page this method is executed, and its result is inserted into the document. This is done using an XSP tag for evaluating expressions:

```
<xsp:expr>count()</xsp:expr>
```

The example in Fig. 4-7 follows exactly that scenario. In Fig. 4-8 a page is shown, which does the same, but the whole Java code is hidden inside a counter library. All pages in the TTS are made using libraries due to the requirement of enabling a non-Java developer to maintain the system. In fact there is not a single line of Java code in Fig. 4-8.

```
<?xml version="1.0"?>

<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="page-html.xsl" type="text/xsl"?>

<xsp:page
language="java"
xmlns:xsp="http://www.apache.org/1999/XSP/Core"
>

<xsp:logic>
    static private int counter = 0;

    private synchronized int count() {
        return counter++;
    }
</xsp:logic>

<page>
    <title>Simple XSP Page</title>
    <p>
            I've been requested
            <xsp:expr>count()</xsp:expr>
            times.
        </p>
</page>

</xsp:page>
```

Fig. 4-7 Simple XSP page source code.

```
<?xml version="1.0"?>

<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="page-html.xsl" type="text/xsl"?>

<xsp:page
language="java"
xmlns:xsp="http://www.apache.org/1999/XSP/Core"
xmlns:counter="http://www.tts.desy.de"
>
<page>
      <title>Simple XSP Page</title>
      <p>
               I've been requested <counter:count/> times.
           </p>
</page>

</xsp:page>
```

Fig. 4-8 The same XSP page as in Fig. 4-7 with usage of a counter library.

4.2.4 SQL and JDBC

The JDBC API is a Java API for accessing virtually any kind of tabular data. The JDBC API consists of a set of classes and interfaces written in the Java programming language that provide a standard API for tool/database developers and makes it possible to write industrial strength database applications using an all-Java API.

The JDBC API makes it easy to send SQL statements to relational database systems and supports all dialects of SQL. But the JDBC 2.0 API goes beyond SQL, also making it possible to interact with other kinds of data sources, such as files containing tabular data.

The value of the JDBC API is that an application can access virtually any data source and run on any platform with a Java Virtual Machine. In other words, with the JDBC API, it is not necessary to write one program to access a Sybase database, another program to access an Oracle database, another program to access an IBM DB2 database, and so on. One can write a single program using the JDBC API, and the program will be able to send SQL or other statements to the appropriate data source.

The JDBC 2.0 API extends what can be done with the Java platform. For example, the JDBC API makes it possible to publish a web page containing an applet that uses information obtained from a remote data source. Or an enterprise can use the JDBC API to connect all its employees (even if they are using a conglomeration of Windows, Macintosh, and UNIX machines) to one or more internal databases via an intranet.

JDBC technology-based driver ("JDBC driver") makes it possible to do three things:

- Establish a connection with a data source.
- Send queries and update statements to the data source.
- Process the results.

Fig. 4-9 gives a simple example of these three steps:

```
Connection con = DriverManager.getConnection(
"jdbc:myDriver:wombat",
      "myLogin",
      "myPassword"
      );
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
      "SELECT a, b, c FROM Table1"
      );
while (rs.next())
{
int x = rs.getInt("a");
String s = rs.getString("b");
float f = rs.getFloat("c");
}
```

Fig. 4-9 Example of accessing a database via JDBC driver.

4.3 Choice of scopes

4.3.1 First implementation cycle

Because TTS is build using new technologies first implementation cycle will contain the smallest set of features. What should be done first is to implement the database model because it is rather complex. There is also no way to reach some functionality level without implementing the whole lifecycle of a process ticket. First cycle should be closed as soon as possible to early get feedback from the users.

Use Case	Implemented Scope
Request Service	Form where hotline operator can register a ticket.
Dispatch Ticket	Full scope.
Search Tasks	Only form where logged in expert can search his tickets.
Update Ticket	Full scope.
Notify Client	Left to be implemented in second cycle.

Fig. 4-10 Functionality chosen to be implemented in first cycle.

Chosen fragments of use cases together consists the core functionality of the system. The implementation will be quick and after it there will be a chance of showing the system to the users and validate.

Form	Function Scope
AddProcessActivity	Adding new activity to an existing process ticket.
AddProcessTicket	Adding (registering) a new process ticket.
CloseTicket	Closing an open ticket and all its activities.
DeleteActivity	Deleting an activity.
EditProcessActivity	Editing (updating) a process activity.
EditProcessTicket	Editing (updating) a process ticket.
Login	Logging into the TTS.
ViewSingleTicket	Viewing (inspecting) one process ticket.
ViewWorkerTickets	Viewing all open tickets, which belongs to logged expert.

Fig. 4-11 Forms aimed to realize first implementation cycle functionality.

4.3.2 Second implementation cycle

TTS is not a huge system; therefore two implementation cycles should be sufficient to reach the full functionality level. Although it is possible that after closing second cycle there will be more new requirements to implement.

Following features should be added:

Use Case	Implemented Scope
Request Service	Form where client can register a process ticket.
Dispatch Ticket	Already implemented.
Search Tasks	Advanced mechanism to search tasks among tickets.
Update Ticket	Already implemented.
Notify Client	Full scope.

Fig. 4-12 Functionality chosen to be implemented in second cycle.

After implementing selected features TTS will reach its full functionality. It will be also possible to get feedback from some potential clients. There will be probably a need to make some small improvements and corrections, after which the user training in full scope can start.

To realize the rest of TTS functionality following forms should be implemented:

Form	Function Scope
SearchTickets	Form for searching tickets according to chosen criteria.
SendEmail	Sending information email to a client.
ClientViewTickets	Limited information record about a ticket for a client.

Fig. 4-13 Forms aimed to realize second implementation cycle functionality.

In second implementation cycle also are planned some forms for generating and viewing statistics for the IPP supervisor. Also administration tools for making the configuration (setting process tickets parameters) easier than using standard database access methods. Exact requirements for these features are already now known.

5 Results

5.1 Overview of achieved solution

The last TTS configuration consisted of installed Sun JDK version 1.3.0.0.2, Tomcat version 3.1.1 and Cocoon version 1.8.2. There was Cocoon version 2 but it was under development and available only via CVS. Cocoon was integrated into Tomcat and TTS was added to it. The thin JDBC driver was installed to work with Oracle RDBMS version 8.1.6.2.0, also the Cocoon's connection pooling was configured.

Finally after closing first implementation cycle there was:

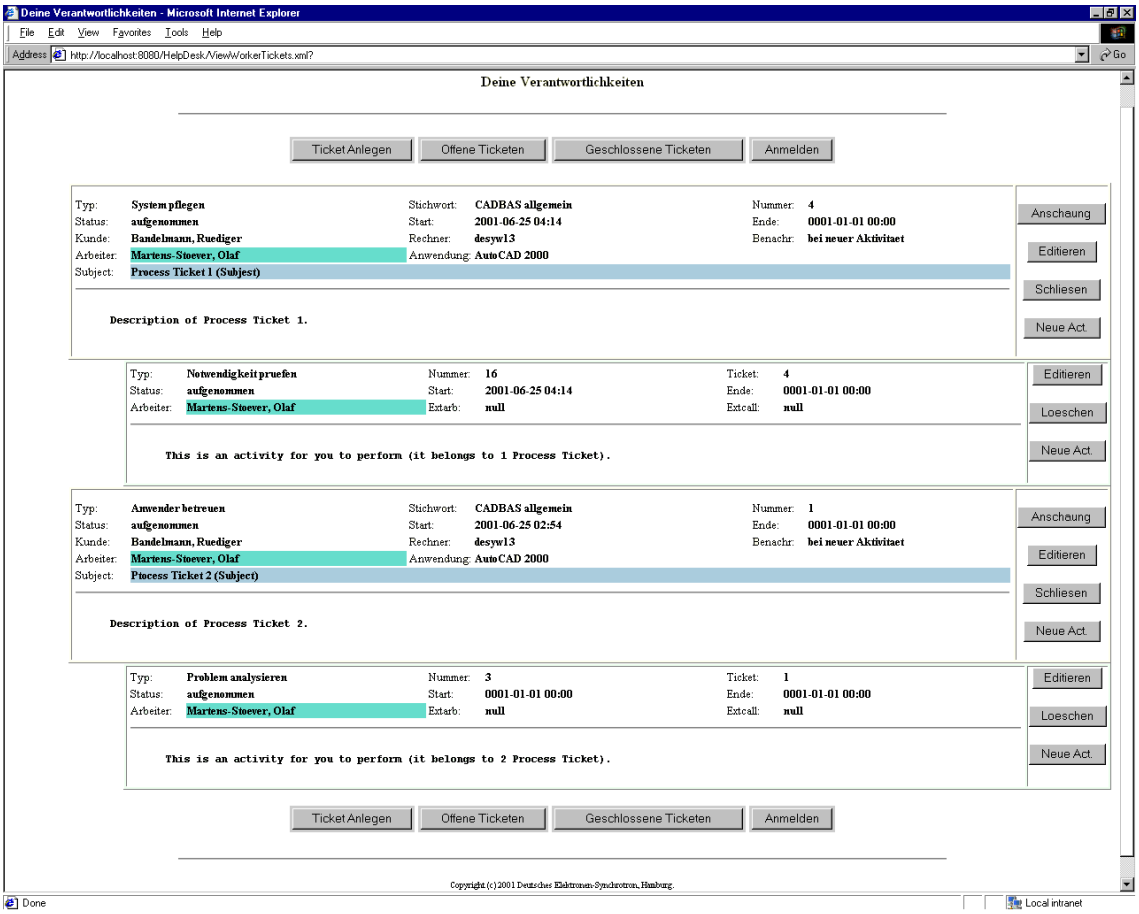
Files quantity	Type of file	Total lines count
12	XSP page	1063
7	XSP library (logicsheet)	1035
11	XSL stylesheet	1198
4	XSL support library	694
3	SQL scripts	222
total: **37**		total: **4212**

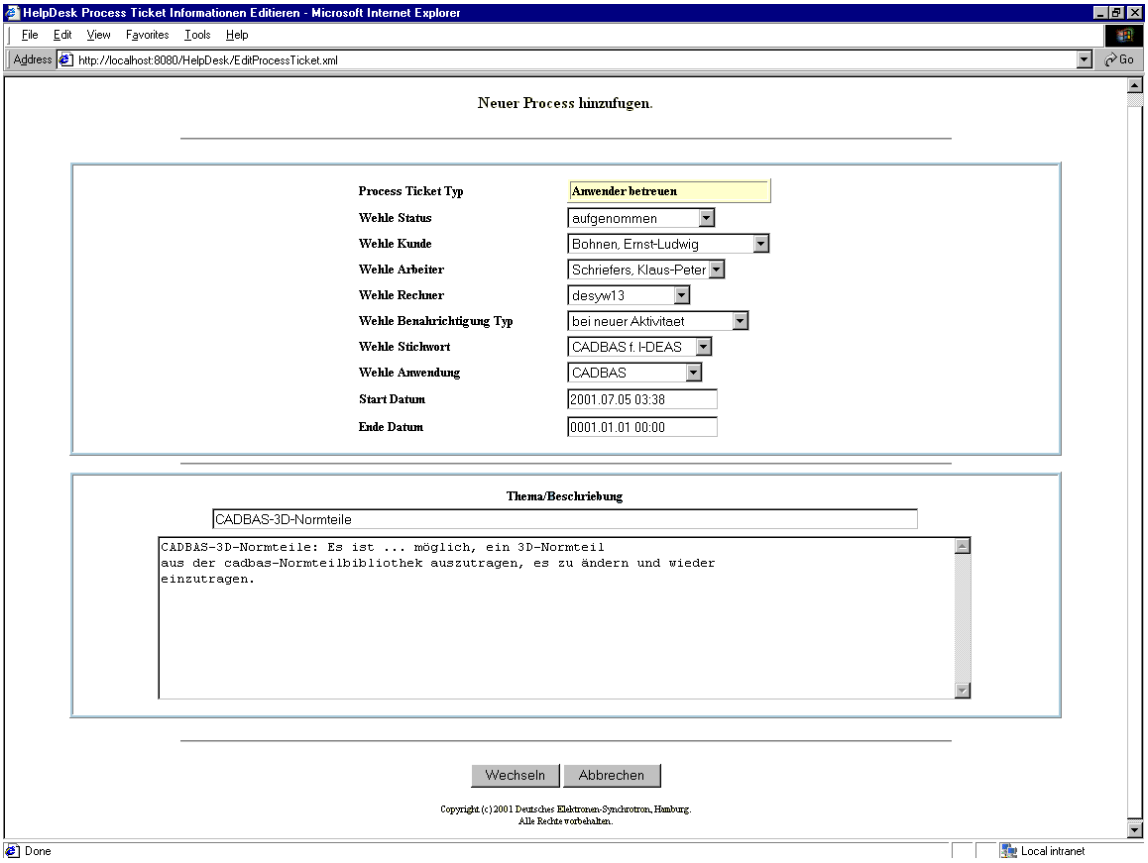
Fig. 5-1 TTS code lines summary.

Originally TTS was operating on a Windows NT workstation, and then shifted to a SPARC machine with Sun Solaris operating system. Additionally TTS was successfully working with Sun's JDK version 1.4 beta, on Windows NT machine. Apache web server was not installed, because there was no production installation, it was possible because Tomcat has an ability to operate as stand alone server for testing. The database structure was created on two Oracle accounts, one for TTS running on the Windows NT and second for TTS on Sun Solaris. For both accounts the default data were provided.
System was very carefully implemented, several fragments of code where changed many times or even eliminated. Compiling the XSP pages results in **12646** lines of Java code, which implemented the producers. The SQL script for inserting default data into the database was 1091 lines long.

Below are two screenshots. The first one is a page where the worker is able to read information concerning all his responsibilities. Originally process tickets

are yellow activities are green. A process ticket was shown to a worker only if he was responsible for the whole open ticket or at least one open activity within it. The second screenshot is a page for registering new process tickets. The goal of designing user interface was to make it as simple as possible. The pages are very clean, there was putted a pressure to make the same thing look the same in different places in the system. Therefore the template, which was responsible for formatting process tickets and activities was moved to a separate file and included in different stylesheets whenever, needed.





There were written two manuals: “TTS Administrator Manual” and “TTS User Manual”. The administrator manual was explaining with details how to install TTS on a new server. How to manage user accounts. How to configure process tickets, activities, statuses, roles, worker access rights and so on. It was also explaining the structure of XSP page, and with details and examples the library interfaces. There was also information where to search solution in case of errors. The user manual was explaining what are processes and activities in TTS, how to log into the system, how to register a new process ticket, how to manage tickets and activities, how to search task to perform and so on.

5.2 Experience

Developing TTS results in a huge amount of experience. XML, XSL, XSP and COCOON were all new technologies. Also building firsts prototypes gives experience.

Used iterative method introduced order in the implementation process, but also has some limitations. Using any method requires lots of experience and discipline. However the second one is up to the developer the first can be only gained in a long time. When the work on TTS started, XML was in DESY completely unknown technology. Therefore scheduling was rather a guessing. Also the architecture design was difficult because it was hard to say what is exactly possible with XML and where in the system to use it. The time for

implementation was three times longer than scheduled. Lots of example programs were developed on purpose to only get an idea what and how can be done. The general conclusion is that a method will work very well in case of developing one more product, which is similar to earlier projects. Is such a situation everything could be exactly planned and standardized. On the contrary when there is a new type of problems to solve, or a completely new technology, first there should be a simple hacking done. The program made in this way should cover a narrow scope of the system functionality, and after reaching a certain stage it should be thrown away. The gained experience allows then to introduce a suitable software development method.

What else could be done? There are some ideas of making the system build out of components. An example component could be a button, a combo box, a text area, a process ticket or an activity. In such a system the XML could be used to describe of which components a page consists. There is another possibility to use XSLT inside components to get their representation depending on some surrounding conditions. It is obvious that this variant will be much more difficult o implement, although the hardest part will be to correctly identify the components.

6 References

Books:

[1] Grady Booch, James Rumbaugh and Ivar Jacobson **"The Unified Modeling Language User Guide"** ADDISON-WESLEY

[2] Terry Quatrani **"Visual Modeling with Rational Rose and UML."** ADDISON-WESLEY

[3] Brett McLaughlin **"Java and XML"** O'REILLY & Associates, Inc.

[4] Jason Hunter and William Crawford **"Java Servlet Programming"** O'REILLY & Associates, Inc.

[5] George Reese **"Database Programming with JDBC and Java"** O'REILLY & Associates, Inc.

[6] David Flanagan **"Java Examples in a Nutshell, Second Edition"** O'REILLY & Associates, Inc.

[7] David Flanagan **"Java in Nutshell, Second Edition"** O'REILLY & Associates, Inc.

[8] Bruce Eckel **"Thinking in Java, Second Edition"** Prentice Hall

[9] Bruce Eckel **"Thinking in Patterns with Java"** (electronic version)

[10] Andres S. Tannenbaum **"Computer Networks"** Prentice Hall

[11] Robert Eckstein **"XML Leksykon Kieszonkowy"** Wydawnictwo Helion

Internet sources:

[12] http://java.sun.com

[13] http://xml.apache.org

[14] http://xml.apache.org/cocoon/index.html

[15] http://jakarta.apache.org/tomcat/index.html

[16] http://jakarta.apache.org/turbine/index.html

[17] http://www.w3c.org/DOM

[18] http://www.w3c.org/MarkUp

[19] http://www.w3c.org/XML

[20] http://www.w3c.org/Style/XSL

[21] http://www.w3c.org/TR/xpath

[22] http://www.w3c.org/XML/Linking

[23] http://www.w3c.org/XML/Schema

[24] http://www.bruceeckel.com

[25] http://www.rz.tu-ilmenau.de/~skoerner/HTML-Taglist

[26] http://webopedia.internet.com

7 Technical appendixes

7.1 Glossary

API **A**pplication **P**rogram **I**nterface
The interface (calling conventions) by which an application program accesses operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code.

Class The prototype for an object in an object-oriented language; analogous to a derived type in a procedural language. A class may also be considered to be a set of objects which share a common structure and behaviour. The structure of a class is determined by the class variables which represent the state of an object of that class and the behaviour is given by a set of methods associated with the class.
Classes are related in a class hierarchy. One class may be a specialisation (a "subclass") of another (one of its "superclasses") or it may be composed of other classes or it may use other classes in a client-server relationship. A class may be an abstract class or a concrete class.

Class Diagram
A view or picture of some or all of the classes in a model.

DOM **D**ocument **O**bject **M**odel
A W3C specification for application program interfaces for accessing the content of HTML and XML documents.

Framework In object-oriented systems, a set of classes that embodies an abstract design for solutions to a number of related problems.

HTTP **H**yper**T**ext **T**ransfer **P**rotocol
A protocol used to request and transmit files, especially webpages and webpage components, over the Internet or other computer network.

Java A high-level programming language developed by Sun Microsystems. Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors. Java source code files (files with a .java extension) are compiled into a format called bytecode (files with a .class extension), which can then be executed by a Java

interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines (VMs), exist for most operating systems, including UNIX, the Macintosh OS, and Windows. Bytecode can also be converted directly into machine language instructions by a just-in-time compiler (JIT).

Java servlet Servlets are server extensions that are written in Java and are associated with particular URLs. When a request for the URL of a servlet is received from a Web browser, the web server invokes the servlet to process the request. Web server provides the servlet with all the information it needs to process the request. It also provides a mechanism for the servlet to send response information back to the web browser. The Servlet API is used to develop servlets. Servlets can be preloaded by web server or loaded on the fly as they are needed

RDBMS **Relational Database Management System**
A type of database management system (DBMS) that stores data in the form of related tables. Relational databases are powerful because they require few assumptions about how data is related or how it will be extracted from the database. As a result, the same database can be viewed in many different ways. An important feature of relational systems is that a single database can be spread across several tables. This differs from flat-file databases, in which each database is self-contained in a single table.

Scenario An instance of a use case – it is one path through the flow of events for the use case.

Sequence Diagram
A diagram that depicts object interactions arranged in time sequence.

SQL **Structured Query Language**
An industry-standard language for creating, updating and, querying relational database management systems. SQL was developed by IBM in the 1970s for use in System R. It is the de facto standard as well as being an ISO and ANSI standard. It is often embedded in general purpose programming languages.

UML **Unified Modeling Language**
A non-proprietary, third generation modelling language. The Unified Modeling Language is an open method used to specify, visualise, construct and document the artifacts of an object-oriented software-intensive system under development. The UML represents a compilation of "best engineering practices" which have proven successful in modelling large, complex systems. UML succeeds the concepts of Booch, OMT and OOSE by fusing them into a single, common and widely usable modelling language. UML aims to be a standard modelling language which can model concurrent and distributed systems.

URL **Uniform Resource Locator**

A standard way of specifying the location of an object, typically a web page, on the Internet. URLs are the form of address used on the World-Wide Web.

W3C **World Wide Web Consortium**

The main standards body for the World-Wide Web. W3C works with the global community to establish international standards for client and server protocols that enable on-line commerce and communications on the Internet. It also produces reference software.

XML **EXtensible Markup Language**

A metalanguage written in SGML that allows one to design a markup language, used to allow for the easy interchange of documents on the World Wide Web.

XSL **EXtensible Stylesheet Language**

A specification for separating style from content when creating HTML or XML pages. The specifications work much like templates, allowing designers to apply single style documents to multiple pages. XSL is the second style specification to be offered by the World Wide Web Consortium (W3C)(www.w3c.org). The first, called Cascading Style Sheets (CSS), is similar to XSL but does not include two major XSL's innovations -- allowing developers to dictate the way Web pages are printed, and specifications allowing one to transfer XML documents across different applications. W3C released the first draft of XSL in August 1998, and promotes the specifications as helpful to the Web's speed, accessibility, and maintenance.

XSLT **EXtensible Stylesheet Language Transformations**

XSLT is the language used in XSL style sheets to transform XML documents into other XML documents. An XSL processor reads the XML document and follows the instructions in the XSL style sheet, then it outputs a new XML document or XML-document fragment. Not all companies use the exact same programs, applications and computer systems. XSLT Recommendation was written and developed by the XSL Working Group and became ratified by the W3C on November 16, 1999.

XSP **EXtensible Server Pages**

The XSP language is a core technology of Cocoon, XML-based Web publishing in Java and one of the seven parts of the Apache XML Project. XSP is used to build dynamic XML content. It was originally created to allow Web authors to generate dynamic content without forcing them to learn a programming language. Because a Web document's content, style and logic are often created by different working groups or individuals, Cocoon aims for a complete separation of the three layers. Using XSP, content, style and logic are separated into different XML files using an XML DTD and are merged using XSL transformation capabilities

7.2 Source code samples

Below (Fig. 7-1) is the source code of the "CloseTicket.xml", XSP page. Inside this page business logic is executed. There is no Java code, because it is hidden inside libraries, and only public tags of these libraries are used. Every tag, which is not a part of a library, is inserted inside the XMLDocument, which is prepared here and destined to be the XML used by stylesheet transformation. The stylesheet is presented below; also there is an example of a XSP library.

```
<?xml version="1.0"?>

<?xml-logicsheet href="/WEB-INF/taglibs/pool.xsp.xsl"?>
<?xml-logicsheet href="/WEB-INF/taglibs/check.xsp.xsl"?>
<?xml-logicsheet href="/WEB-INF/taglibs/cbox.xsp.xsl"?>
<?xml-logicsheet href="/WEB-INF/taglibs/security.xsp.xsl"?>
<?xml-logicsheet href="/WEB-INF/taglibs/params.xsp.xsl"?>
<?xml-logicsheet href="/WEB-INF/taglibs/context.xsp.xsl"?>
<?xml-stylesheet type="text/xsl" href="CloseTicket.xsl"?>

<?cocoon-process type="xsp"?>
<?cocoon-process type="xslt"?>

<xsp:page
  xmlns:xsp="http://www.apache.org/1999/XSP/Core"
  xmlns:esql="http://apache.org/cocoon/SQL/v2"
  xmlns:request="http://www.apache.org/1999/XSP/Request"
  xmlns:util="http://www.apache.org/1999/XSP/Util"
  xmlns:response="http://www.apache.org/1999/XSP/Response"
  xmlns:session="http://www.apache.org/1999/XSP/Session"
  xmlns:check="http://www.helpdesk.desy.de/check"
  xmlns:cbox="http://www.helpdesk.desy.de/cbox"
  xmlns:security="http://www.helpdesk.desy.de/security"
  xmlns:params="http://www.helpdesk.desy.de/params"
  xmlns:context="http://www.helpdesk.desy.de/context"
  create-session="true"
>
<page>
       <check:initialize/>
       <params:initialize/>

       <security:roles>
              <security:role>benutzer</security:role>
              <security:role>verwalter</security:role>
       </security:roles>

       <params:paramset>
              <params:param>TICKETID</params:param>
       </params:paramset>
       <params:true>

              <check:isnotnull param="CloseTicketClose" scope="closebutton"/>
              <!-- make ticket and all its activities closed -->
              <check:true scope="closebutton">
                     <!-- close ticket -->
                     <esql:connection>
                     <esql:pool>HelpDesk</esql:pool>
                     <esql:execute-query>
                            <esql:query>
                                   UPDATE HelpDesk_ProcessTickets SET StatusID = 2
                                   WHERE ID =
                                   <esql:parameter>
                                          <request:get-parameter name="TICKETID"/>
                                   </esql:parameter>
                            </esql:query>
                     </esql:execute-query>
                     </esql:connection>
                     <!-- close tickets activities -->
                     <esql:connection>
                     <esql:pool>HelpDesk</esql:pool>
```

```
                        <esql:execute-query>
                                <esql:query>
                                        UPDATE HelpDesk_ProcessActivities SET StatusID = 2
                                        WHERE ProcessTicketID =
                                        <esql:parameter>
                                                <request:get-parameter name="TICKETID"/>
                                        </esql:parameter>
                                </esql:query>
                        </esql:execute-query>
                        </esql:connection>
                        <!-- redirect to the appropriate place -->
                        <response:send-redirect
                         location="/HelpDesk/ContextDispatcher.xml"/>
                </check:true>

                <check:false scope="closebutton">
                        <esql:connection>
                        <esql:pool>HelpDesk</esql:pool>
                        <esql:execute-query>
                                <esql:query>
                                        SELECT * FROM helpdesk_vprocesstickets WHERE ID =
                                        <esql:parameter>
                                                <request:get-parameter name="TICKETID"/>
                                        </esql:parameter>
                                        ORDER BY ID DESC
                                </esql:query>
                                <esql:results>
                                <esql:row-results>
                                        <TICKET><esql:get-columns/></TICKET>
                                </esql:row-results>
                                </esql:results>
                        </esql:execute-query>
                        </esql:connection>
                </check:false>
        </params:true>
</page>
</xsp:page>
```

Fig. 7-1 Source code of CloseTicket.xml XSP page.

Below (Fig. 7-2) the stylesheet for CloseTicket.xml XSP page is presented. The task of it is to translate the XMLDocument generated inside CloseTicket.xml into an html, which is then send to the user web browser. This stylesheet includes other library stylesheets, which are used to transform ticket information, colours and so on.

```
<?xml version="1.0"?>

<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
>

<xsl:include href="WEB-INF/stylesheets/style.xsl"/>
<xsl:include href="WEB-INF/stylesheets/cbox.xsl"/>
<xsl:include href="WEB-INF/stylesheets/error.xsl"/>
<xsl:include href="WEB-INF/stylesheets/tickets.xsl"/>

<xsl:template match="/page">
        <html>
        <head>
        <title>Willst du diser Process und alle ihre Activiteten schlisen?</title>
        </head>
        <body BGCOLOR="{$bgcolor}">
                <xsl:choose>
                        <xsl:when test="//error">
                                <xsl:call-template name="errors"/>
                        </xsl:when>
                        <xsl:otherwise>
                                <xsl:call-template name="header"/>
                                <xsl:call-template name="table"/>
                                <xsl:call-template name="footer"/>
```

```
                        <xsl:call-template name="copy-right"/>
                    </xsl:otherwise>
                </xsl:choose>
        </body>
        </html>
</xsl:template>


<xsl:template name="header">
        <br/>
        <h3 align="CENTER">
                Willst du diser Process und alle ihre Activiteten schlisen?
        </h3>
        <hr width="70%"/>
    <br/>
</xsl:template>

<xsl:template name="table">
        <xsl:call-template name="TICKETS">
                <xsl:with-param name="ticketbuttons">yes</xsl:with-param>
                <xsl:with-param name="activitybuttons">no</xsl:with-param>
                <xsl:with-param name="ticketdescription">yes</xsl:with-param>
                <xsl:with-param name="activitydescription">no</xsl:with-param>
                <xsl:with-param name="activities">no</xsl:with-param>
                <xsl:with-param name="userid"></xsl:with-param>
        </xsl:call-template>
</xsl:template>

<xsl:template name="footer">
        <br/><hr width="70%"/><br/>

        <table align="center">
        <tr>
        <td>
        <form action="/HelpDesk/CloseTicket.xml" method="POST">
        <xsl:element name="input">
                        <xsl:attribute name="type">hidden</xsl:attribute>
                        <xsl:attribute name="name">TICKETID</xsl:attribute>
                        <xsl:attribute name="value">
                                <xsl:value-of select="/page/TICKET/ID"/>
                        </xsl:attribute>
                </xsl:element>
                <input type="submit" name="CloseTicketClose" value="Schliesen"/>
        </form>
        </td>
        <td>
        <form action="/HelpDesk/ContextDispatcher.xml" method="POST">
                <input type="submit" name="x" value="Abbrechen"/>
        </form>
        </td>
        </tr>
        </table>
</xsl:template>

</xsl:stylesheet>
```

Fig. 7-2 Stylesheet for CloseTicket.xml XSP page.

Below (Fig. 7-3) an example of a XSP library is presented. A library is a XSL stylesheet (called a logicsheet because of its purpose), which is used to replace certain tags inside the source document with Java code. Therefore it is present inside this stylesheet. This code is then inserted into the producer generated from the XSP page, which uses this logicsheet. Depending on the implementation of a library it can be considered as a reusable component. All libraries made for TTS are rather general purpose ones, and can be used in other Cocoon system.

```
<?xml version="1.0"?>

<xsl:stylesheet
        version="1.0"
```

```
	xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
	xmlns:xsp="http://www.apache.org/1999/XSP/Core"
	xmlns:params="http://www.helpdesk.desy.de/params"
>

<xsl:template match="xsp:page">
<xsp:page>
	<xsl:apply-templates select="@*"/>
	<xsp:structure>
		<xsp:include>java.util.Hashtable</xsp:include>
	</xsp:structure>

	<xsp:logic>
	// if all parameters are present
		// then this variable will be set to true
		boolean Request_Page_Parameters;
	</xsp:logic>
	<xsl:apply-templates/>
</xsp:page>
</xsl:template>

<xsl:template match="params:paramset">
	<xsp:logic>
		Request_Page_Parameters = true;
	</xsp:logic>
	<params>
	<xsl:for-each select="params:param">
	<xsp:logic>
		if(request.getParameter("<xsl:value-of select="."/>")==null)
		{
			Request_Page_Parameters = false;
			<xsp:content>
			<error>
				Diese Seite soll nicht unmittelbar zugrifft werden.
			</error>
			</xsp:content>
		}
		else
		{
			// this is necessary
			// because the element name is
			// not known now
			xspParentNode = xspCurrentNode;
			xspNodeStack.push(xspParentNode);
			xspCurrentNode =
				document.createElement("<xsl:value-of select="."/>");

			xspParentNode.appendChild(xspCurrentNode);
			xspCurrentNode.appendChild(

			document.createTextNode(
				request.getParameter("<xsl:value-of select="."/>")));
			((Element) xspCurrentNode).normalize();
			xspCurrentNode = (Node) xspNodeStack.pop();
		}
	</xsp:logic>
	</xsl:for-each>
	</params>
</xsl:template>

<xsl:template match="params:initialize">
	<xsp:logic>
		Request_Page_Parameters = true;
	</xsp:logic>
</xsl:template>

<xsl:template match="params:true">
	<xsp:logic>
		if(Request_Page_Parameters)
		{
			<xsl:apply-templates/>
		}
	</xsp:logic>
</xsl:template>
```

```
<xsl:template match="params:false">
	<xsp:logic>
		if(!Request_Page_Parameters)
		{
			<xsl:apply-templates/>
		}
	</xsp:logic>
</xsl:template>

<!-- default taglibrary template -->
<xsl:template match="@*|*|text()|processing-instruction()" priority="-1">
	<xsl:copy>
		<xsl:apply-templates select="@*|*|text()|processing-instruction()"/>
	</xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Fig. 7-3 Source code of XSP security library.