SVN @ T-Mobile –
Die standardisierte Entwicklungsumgebung - Konzeption und Umsetzung

SubConf 2008, München, Carsten Sensler & Andre Karalus

Agenda

- Prerequisites
- SOA Backplane in a nutshell
- The development tool chain/ environment
- Our standardized development tool chain

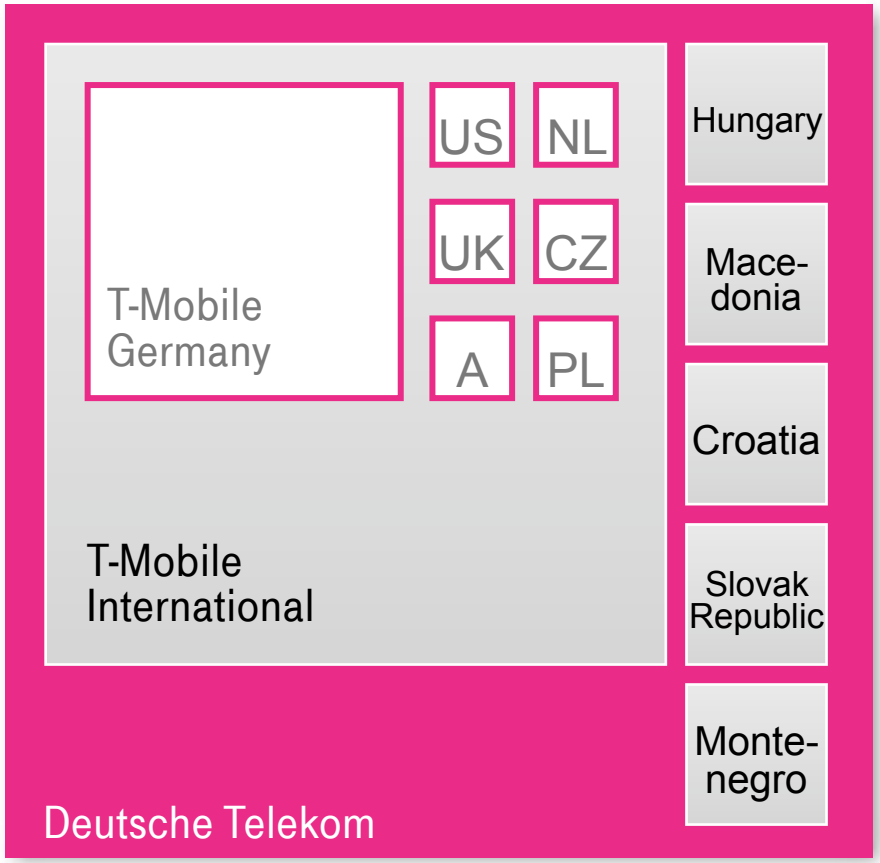
Who we are?

- Dipl.-Ing. Carsten Sensler
  - Employee of T-Mobile Deutschland GmbH since April 2007 (but since December 2005 working in the SOA Backplane program)
  - Department of Enterprise Integration
  - System & Solution Designer
  - Functional leader of the international Service Provisioning Team

- Dipl.-Inf. Andre Karalus
  - Freelancer,
  - since March 2006 consultant for T-Mobile
  - Designer and developer of the runtime core component of the ESB in SOA Backplane and of the Core from the Service Repository

Prerequisites

Corporate Structure.
Subsidiaries and affiliates.



- Direct or indirect investments by Telekom Group in companies dealing with mobile communications in twelve countries
- The T-Mobile brand is represented in Germany, Austria, Hungary, Great Britain, the Czech Republic, the Netherlands, the Slovak Republic, Croatia and the USA
- Almost 125 million customers in the majority holdings

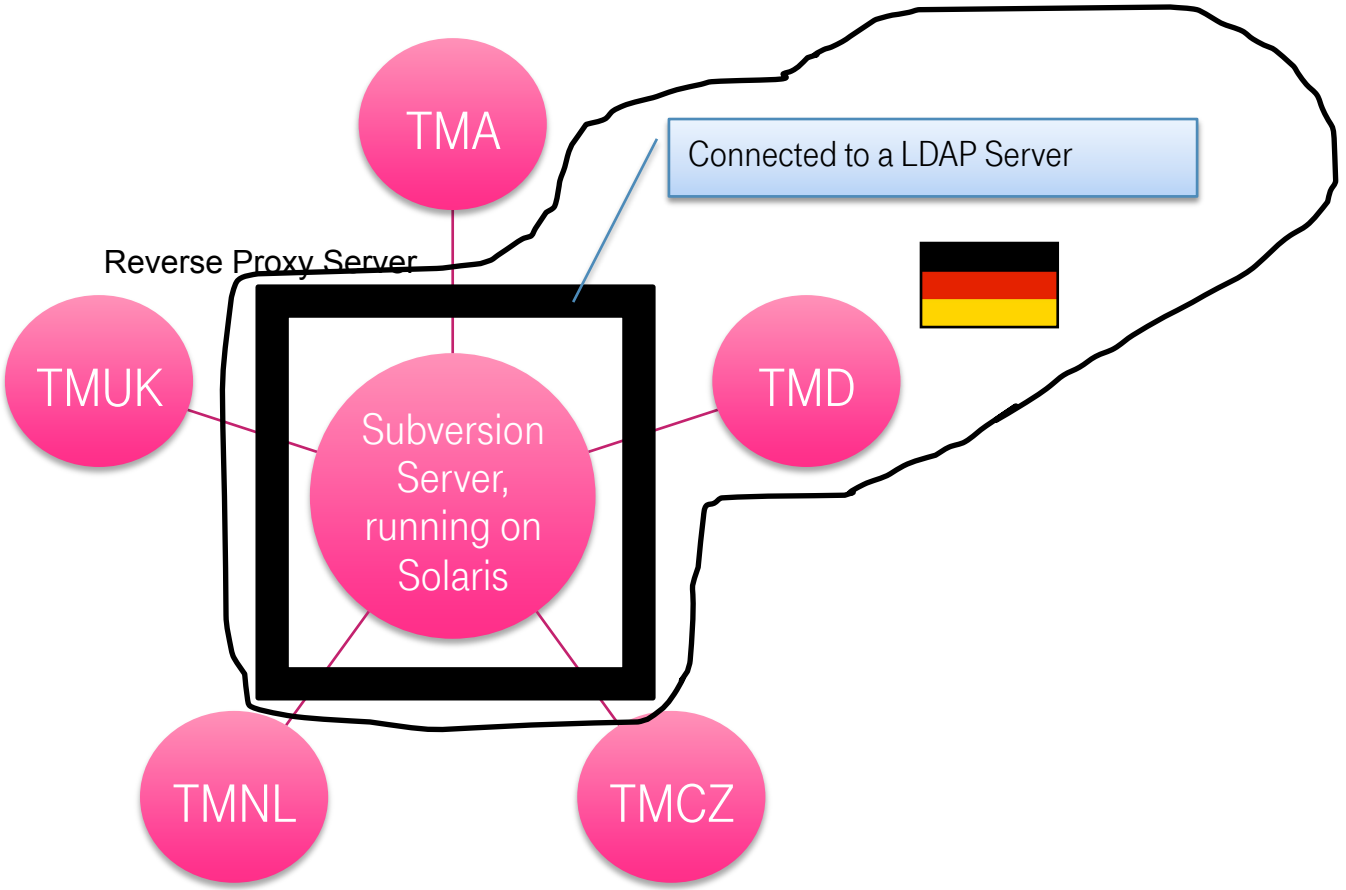
SOA Backplane Participant

Development team/ process/ approach

- Most members of the development team are located in Germany
  - Some members are located in Austria and United Kingdom
  - it consists also of external contractors

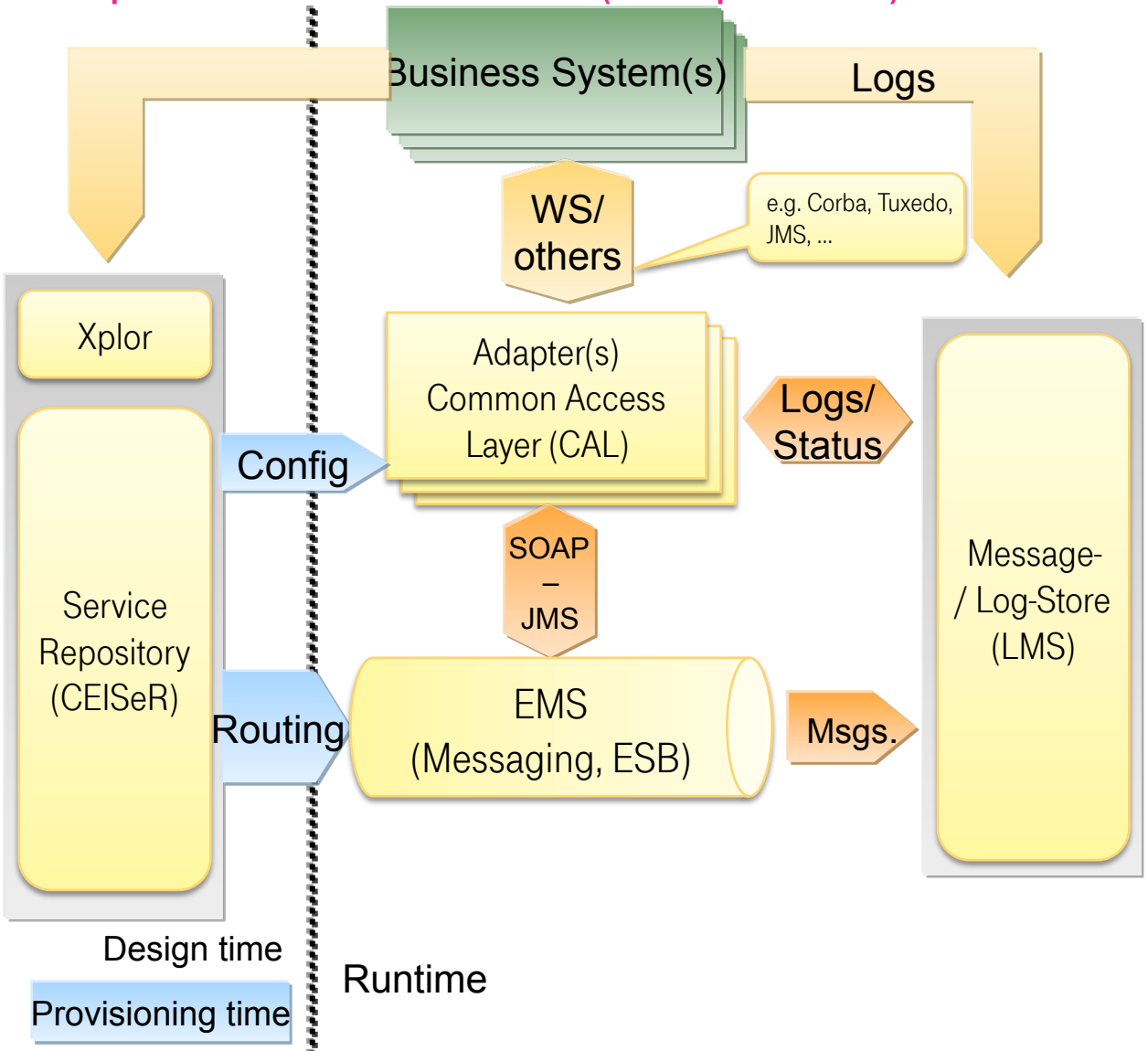
- The development process is very agile
  - especially incremental and iterative
  - it is test-driven
  - We try to use the model driven software approach as far as it makes sense
    - We have a complex build mechanism, because of the heavy utilized MDSD approach
  - The ratio between generated and hand-coded source code is very high
  - We automated the software development and therefore we automated the build and test mechanism, too
- We decided to set the whole development stuff in the way, that it is offline useful and functional

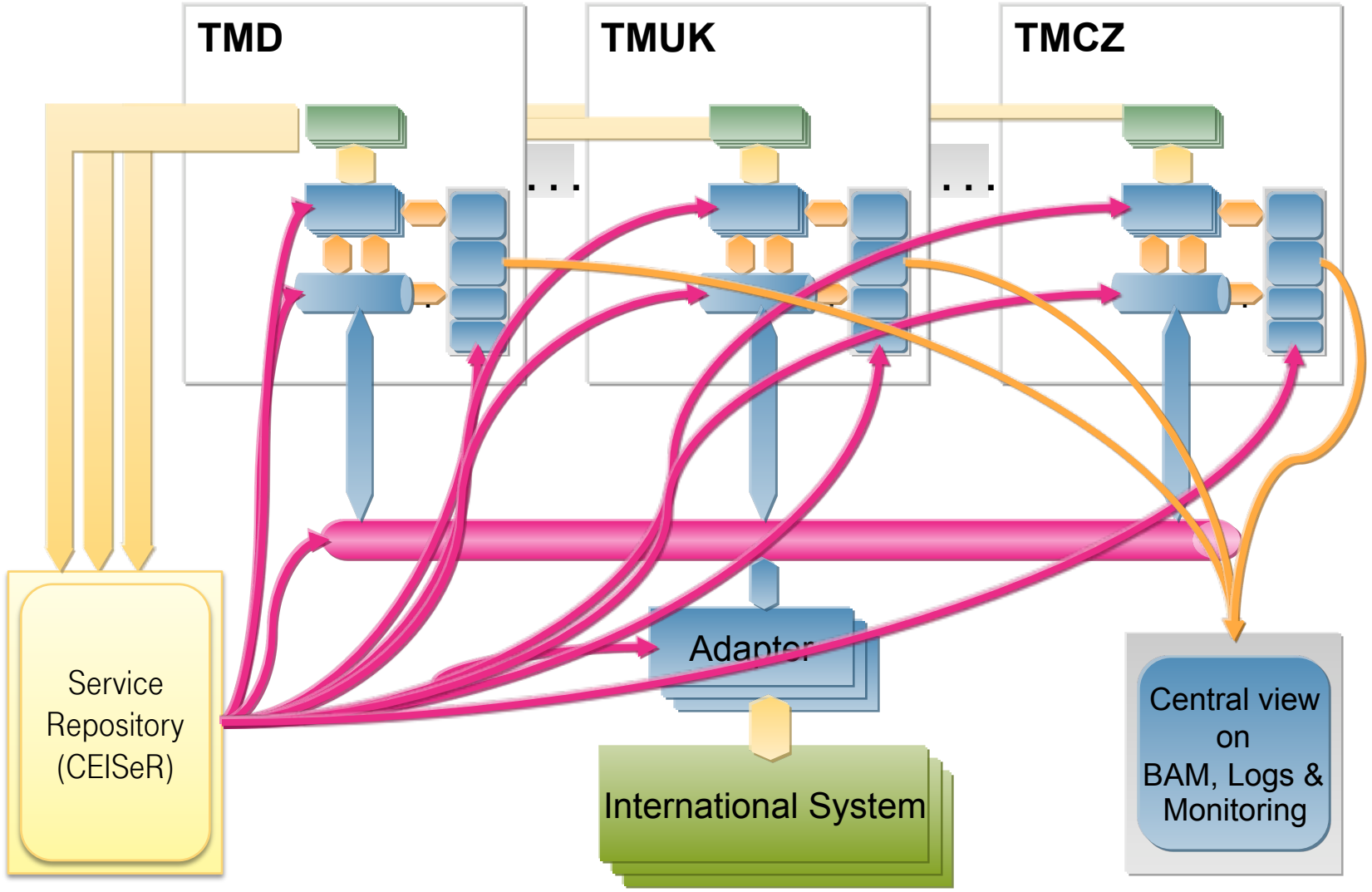
Subversion Infrastructure



SOA Backplane explained on few slides

SOA Backplane – overview (simplified)

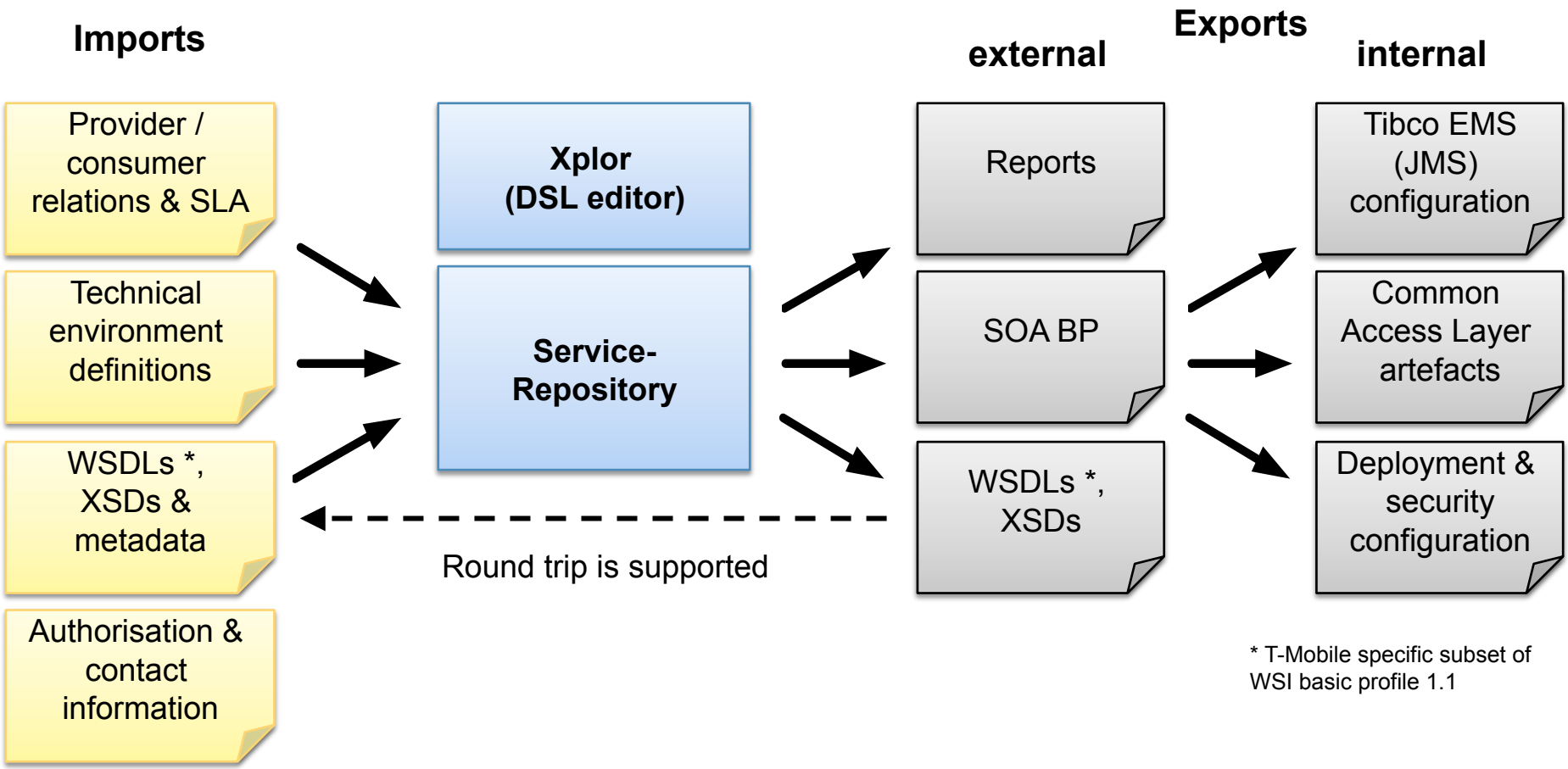



TMD
TMUK
TMCZ
Service Repository (CEISeR)
Adapter
International System
Central view on BAM, Logs & Monitoring

Intention of the SOA Backplane program

- SOA Backplane will deliver a number of software systems and standards, namely
  - a service bus which is the SOA communication infrastructure
    - Service repository
    - Access layer framework
    - Basic messaging infrastructure (JMS)
  - additional value adding components and functionality including
    - logging, monitoring
    - service contract management
    - business activity monitoring
    - transport components for B2B communication
  - the Backplane Guide and SOA Governance as a set of guidelines and rules as to how SOA will be implemented within T-Mobile.

SOA Backplane – Service Repository Interfaces



The Service Repository in a nutshell

- Provide all information needed by service participants for consistent service implementation and utilisation (architecture)
- Store definition of different SOA backplane environments and binding of service participants to these environments (binding)
- Support fully automated configuration of SOA backplane environments (dev, test, prod, ...) (service provisioning)
- Support SOA governance and impact analysis (change / incident) (management)

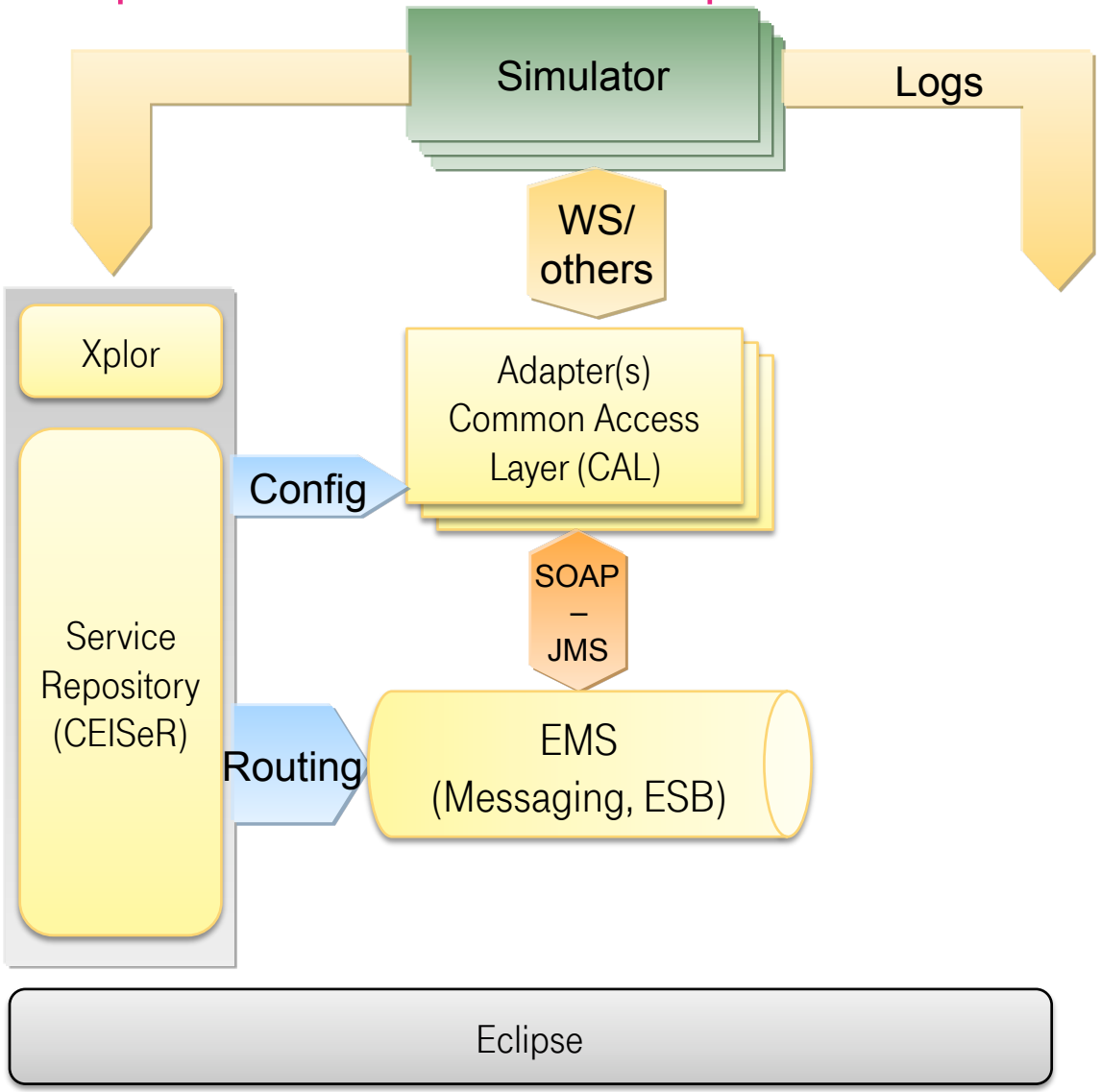
The development tool set/ environment

Development tool set

- Eclipse 3.2
- Ant 1.7
- Groovy
- openArchitectureWare 4.2
- Hibernate 3.2
- Tibco EMS Server 4.5
- JDK 1.5.0_14
- HSQL DB (offline) or Oracle (online)
- JBoss 4.0.4GA
- Several third party open source libraries

- Platform: Linux and Windows

SOA Backplane – local development tool set



The standardized development tool set @ T-Mobile

Why a standardized development tool set?

- Reduce effort and time for enabling new team members
- Increase the effort for supporting own created development environments
- To be able, to get a local SOA Backplane runtime up and running for testing purposes very fast
- To be able, to retest defects or incidents. It doesn't matter, who does the retest and analyse, because every developer has the same environment.

How is the standardized development tool set implemented?

- From the conceptual point of view, it was very important to implement the standardized development tool set in a way, that it is very easy for everybody to get the whole stuff running on their own machines with minimal support

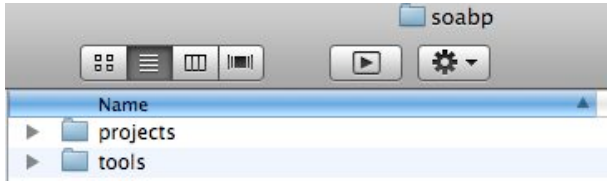
- We took care of the maintenance of the tool set.
  - It was very important to minimize the effort for maintaining the tools set
    - Just check in a new version of a tool into Subversion (Buildmaster) and all users get via “svn update” this integrated new version of the tool or a completely new tool

- Most of the implementation is done via complex ant scripts and shell/cmd scripts. Some minor aspects are implemented with Groovy.

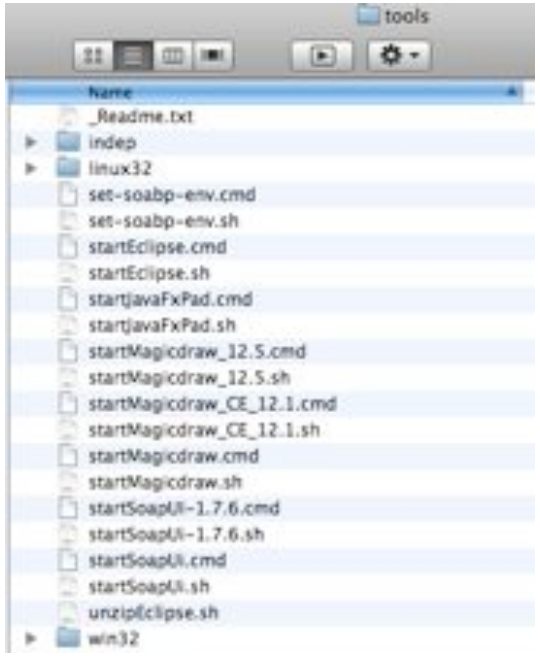
- We lay on a standardized directory structure and two environment variables

Structure of the standardized development tool set - 1

- In general the whole standardized development tool set consists of two directories
  - tools
    - Development tool chain
  - projects
    - Development (Eclipse) projects

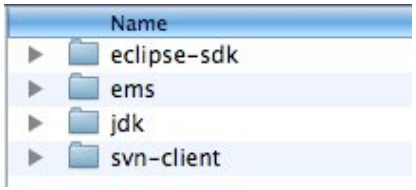


- Tools
  - win32
  - Linux32
  - indep
  - Several shell scripts and cmd files

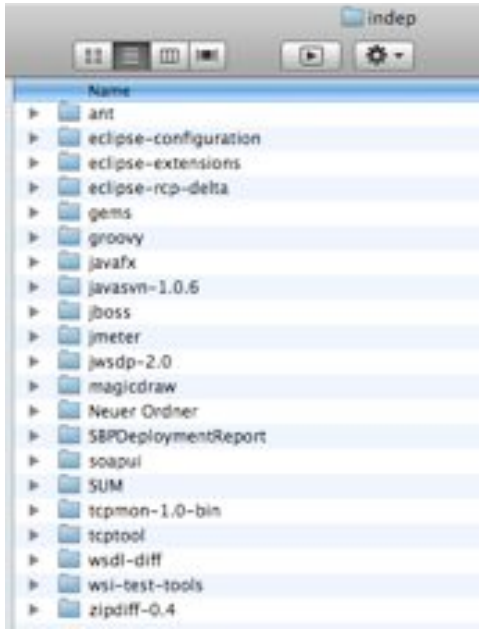


Structure of the standardized development tool set - 2

- win32, linux 32



- indep
  - Platform independent tools



Some restrictions/ constraints/ rules/ more details -1

- We use the shared configuration capability of Eclipse and the Extension Location mechanism
- The tool set exists for Windows and Linux 32bit
- The build process is realized with Ant;
  - We use nearly every available feature (for Java) from Ant and Ant-crontib
  - We implemented some special tasks for Ant by ourselves
- Every development project must have (and implement) several “public” Ant tasks, for example
  - initialize
  - generate (if the project uses a code generator)
  - Compile
  - build.jar
  - build.dist
  - all

Some restrictions/ constraints/ rules/ more details -2

- There are project templates provided that make it easy to initially setup such a project containing the correct Ant build file
- In every assembled JAR file, we store in the MANIFEST.MF file the Subversion revision number, on which the JAR file based on.
- To achieve this, it is part of the build process to look up Subversion metadata for according project and include that into the MANIFEST.MF file

Some restrictions/ constraints/ rules/ more details - 3

- The whole stuff needs 3,5 GB disc space (as a working copy from svn)

- In the project directory exists a project called “_common.used.libs”. This is the only place, where third party libraries should be stored. Everyone can reference this project. We centralized the use of the third party libraries to have a better control

- Another very important project is “_build.management”. This project is the central build management project. It consists only of Ant scripts and property files. There are many macros for Ant defined and stored. It is the overall build project. With a special Ant target, you can generate a development project template

Effort for the realization/ maintenance of the standardized development tool set?

- Realization
  - Round about 25 MD effort for the first realization was needed
- Maintenance
  - Small changes, for example upgrading Ant
    - 0,5 MD
  - Middle complex changes, for example upgrading openArchitctureWare
    - 4 MD
  - Heavy complex changes, for example upgrading Eclipse
    - 8 MD
- The effort includes regression tests of the whole tool chain

Benefit/ advantages of the standardized development tool set - 1

- Since we have this standardized development tool chain in place, everybody (who has a LDAP account) is able to get a full working development environment in round about 2-4 hours (it depends on the location, see slide 8)
- We minimized the effort and support for setting up the development environment extremely
- We work often at different locations and with our standardized development environment, we ensure, that the developer developing with the same software and the test results are more comparable
- We would like to avoid sentences like “works for me”, “but please use this library XYZ”. If the developer did a complete check in, it is nearly almost the case, that the new code works by everybody proper.

Benefit/ advantages of the standardized development tool chain - 2

- We would like to avoid long lasting installations and googleing for the proper version of a tool/ library
- We enable the developers to do a decentralized integration of their work by providing an integration environment on their box thus supporting an early integration not only for build time but for real runtime!
- We are able to retest a defect or an incident with exactly the same development environment, to ensure that we have the same conditions like the assembled and deployed software component.
  - This aspect is very important, because we are developing Eclipse Plugins and a Eclipse based rich client application and therefore we have a direct dependency on the Eclipse distribution
- How long does it take, to get a SOA BP runtime environment (and the design time components) up and running locally?
  - Full checkout – 30 min (depending on the network capability if you reside in other NatCos)
  - Full build (with automated regression tests of the Service Repository and the CAL) – 45 minutes (on a usual developer notebook)
  - Within 1,5 hours, you have a local environment up and running from scratch

Summary

- Fast enabling of new team members
- Fast set up of a local SOA Backplane environment (except the logging and monitoring components and the service provisioning components)
- Easily rebuilding of every JAR file is possible
- Error-less
- Easy to spread the latest versions of the used tools
- Minimal effort for setting up a really complex development tool chain
- Improves defect and incident analyses

Discussion

Thank you.