WERKSTATT UNTERNEHMENSSOFTWARE KARLSRUHE (WUSKAR)

CASE STUDY

USER PROVISIONING PROCESSES IN IDENTITY MANAGEMENT ADDRESSING SAP CAMPUS MANAGEMENT

Marwane El Kharbili
Thomas Mathes
Praharshana Perera

Heiko Schandua
Tomas Stiller
Kim Langer
Christian Emig
Sebastian Abeck

Cooperation & Management (C&M)
Institute for Telematics
Universität Karlsruhe (TH)
wuskar@cm-tm.uka.de

Abstract

This document is the report of the work of an ISWA working team on a "WUSKAR case study". This study tackles on the desire of meta directory synchronisation with a proprietary SAP R/3 system in the context of an identity management system. Early tasks concern identifying exact desires and scenarios, modelling the synchronisation process, identifying what relevant data is to be processed, as well as proposing templates for the matching and transformation process. Intermediate tasks are related to the technical aspects of the case study, as well as problem task division and progress management, regular review of strategic and technical choices.

Keywords

SAP Campus Management, (SAP CM), Meta Directory, Synchronization, LDAP, JNDI, LDIF, SAP Java Connector (JCo), Java, BAPI, WUSKAR, BPMN, Identity Management (IdM)

Learning Goals

1. Learn how to synchronize an enterprise resource planning system with a meta directory.
2. Understand the idea behind directories in this context, especially LDAP-Directories.
3. Obtain knowledge about integration for SAP Systems like BAPI and the JCo.
4. Be able to set up a synchronization process.

Major Sources

[C&M-I-ID] Cooperation & Management: IDENTITY MANAGEMENT IN THE FOCUS OF APPLICATION INTEGRATION, Course Unit of the Lecture INTERNET SYSTEMS AND WEB APPLICATIONS (ISWA), http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck).

[SAP-JCo] Cooperation & Management, Heiko Schandua: Einführung zum SAP Java Connector, Dokument im Rahmen der C&M Technologien und Werkzeuge, Universität Karlsruhe (TH), C&M (Prof. Abeck), Mai 2005.

[EDUPERSON01] Official Homepage of Internet2/Educause eduPerson Working Group http://www.educause.edu/eduperson/

Table of Contents

0 INTRODUCTION 6
0.1 Motivation 6
0.2 Progression of the Case Study 7
1 ANALYSIS 8
1.1 Business Area 8
1.1.1 Scenario 8
1.1.2 Goals 8
1.1.3 Processes 9
1.1.4 Business Objects 11
1.2 System Area 12
1.2.1 Meta Directory 12
1.2.2 Legacy System 19
1.2.3 System Process 23
1.3 Exercises 24
2 DESIGN 26
2.1 Application Architecture 26
2.2 Interface of the Directory 28
2.2.1 LDAP Schema Design 28
2.2.2 LDAP Communication 32
2.3 Interface to the Legacy System 34
2.4 The Synchronization Application 36
2.5 Exercises 39
3 IMPLEMENTATION, DEPLOYMENT AND USAGE 41
3.1 Implementation 41
3.1.1 LDAP Side 41
3.1.2 Synchronizer Side 50
3.1.3 SAP Side 52
3.2 Increment Definition 55
3.3 Deployment 57
3.3.1 Deployment Context 57
3.3.2 LDAP Side 57
3.3.3 SAP Side 59
3.4 Exercises 60
4 OUTLOOK 61
APPENDIX A JAVA SOURCE CODE (JCO / LDAP) 62
APPENDIX B BAPI INTERFACE DESCRIPTION 72
APPENDIX C LDAP PLATFORM INSTALLATION AND CONFIGURATION 73
TABLES 76
Abbreviations and Glossary 76
Index 77
Information and Exercise Slides 77
References 78

0 INTRODUCTION

0.1 Motivation

(1) Identity Management (IdM) as a critical aspect of enterprise systems and B2B processes

(2) Differents components of the system have same users

(3) Users have to be indentified by every component

(4) System must dispose of reliable authentification procedures

(5) Security of the information and its integrity must be guaranteed

(6) Wuskar case study : University Resource Planning system (URP) has to be synchronized with a meta directory

Information 1: Introduction

User Identity Management (IdM) is an everyday concern among every working group whose business processes are supported by an underlying information system, be it a small sized company or a multinational company Users of the company's systems can come from outside or inside the organization. Thus access to the system's information and applications has to be managed efficiently. This access has to take into account the security of the information systems as well as the integrity and exposal of sensitive information. The management of user identities through multiple applications connected to one another sensibly complicates this task. Systems that enable to manage this aspect often include building blocks such as: Password reset, Password synchronization, Single sign-on, Access management software.

This concern is more specifically important to study in the context of EAI systems, because they represent the most frequently occurring instance of heterogeneous systems having to cope with multiple user identities. Our concern in this study is Identity Management, observed in a specific context that is still to be specified.

In a University Resource Planning System (URP), data about students has to be kept up to date with the currently valid credentials for each user. Though, different functional divisions or departments of the university have each their own repository of information. For logical reasons such as too big quantities of data from which just a small portion could be exploitable from other counterparts, as well as for security reasons as some information has to be kept private to a department or division. Should this information be exchanged, for cooperation reasons, for example simply on request from a department, we would have to dispose of a secure process to proceed to the migration of this information from one division to another, in a completely transparent and secure way for overlaying processes.

0.2 Progression of the Case Study

(1) Case study context presentation

(2) Who are the actors and what needs have they?

(3) Business process modeling

(4) Business objects modeling

(5) Application architecture

(6) Synchronization process

(7) Legacy systems and meta-directories

(8) Design and Implementation documentation

Information 2: Study Progress

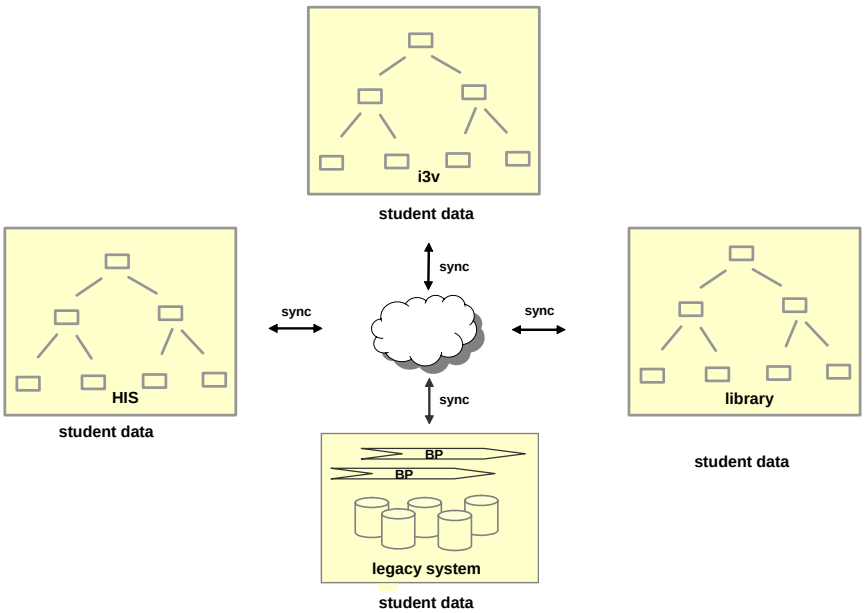
In this case study, we are about to approach a system, which consists of a meta directory and a SAP Campus Management system, coupled together through a synchronisation system. We want to be able to synchronize the meta directory with the SAP system through a reliable process. What a meta directory and what does exactly a synchronization process do, we are about to discuss later in this document. Now that we have an idea about the studied system and the desired functionality we are about to model, we'll want to have the latter described and specified. In order to do it we'll first get familiarized with the context of the case study in a more deepened way. We'll then get to know, who the actors of our system are and have their needs captured. We'll also want to anticipate what possible scenarios will be likely to happen. Furthermore we will identify concepts and processes, particularly the synchronisation process, as well as our business objects. In a further design part we will define our application architecture. We will have the frontiers of the system components defined, then we will elaborate collaboration models for the designed components, and then we'll have the system's internal interfaces presented and explained. We will examine the synchronization process with its compound processes such as the transformation and mapping processes and how they are going to cooperate together. Throughout the document we'll have as well a closer look at SAP-CM and meta directories. Ultimately we will show how all developed concepts can be implemented.

1 ANALYSIS

1.1 Business Area

1.1.1 Scenario

At a university, there are several different organization units, which have many different tasks. To be able to handle these tasks, each of these organisations has an IT infrastructure of her own which meets the requirements of the individual institutions. There for example is a student office which must manage students. The students need the possibility of applying for examinations. Furthermore the results of the examinations must be held tight digitally.
Secondly, there are the library, the canteen, different faculties and the administration. Each of these organisations manages students and saves internally specific data with access rights of these students. Any matriculated student can for example lend books in the library.
Or a student works as a help-scientist on a faculty and has additional access rights to special data.

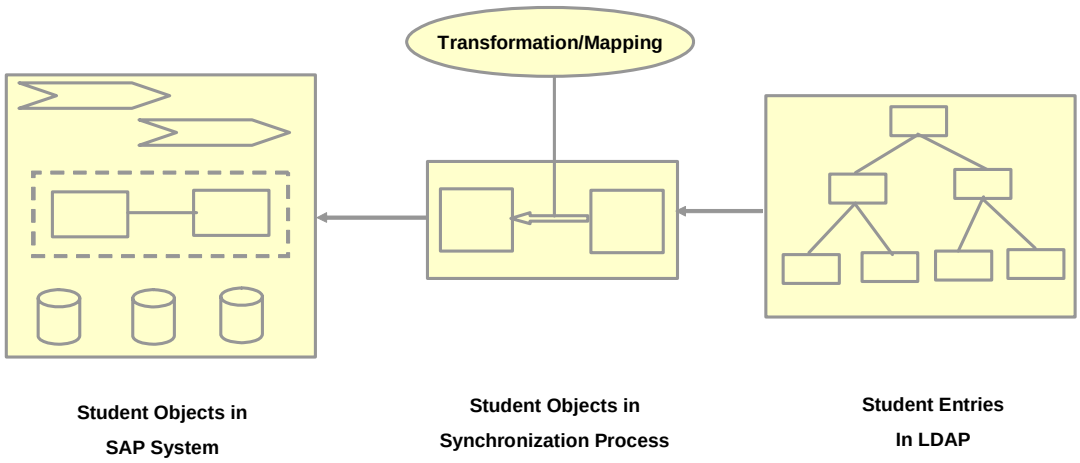


Information 3: Decentralized Synchronization of the Student data

A problem consists in the fact that the data of the students is saved at different places in different software systems. Since the data is saved redundantly, it is not possibly to manage the access rights centrally. Through this it is very difficult to offer the opportunity to the students, to authenticate with the same login data everywhere. To make this possible, the data must be synchronized between the different systems. In this study we describe a part of concept which offers the possibility of enlarging an available IT infrastructure so that the data can simply be synchronized between the different systems.

1.1.2 Goals

The central and primary aim is to introduce a concept, with which it is possible to enlarge and adapt an already available IT infrastructure in a way that IdM and single sign-on is easier to establish. Because of the large size and the high complexity of this formulation, we only look at the synchronization between the directory and legacy system. We primary describe a basic concept, which is necessary for a central IdM infrastructure.



Information 4: Scenario of the Synchronization Process

This concept has three fundamental goals.
First of all a structure for a primary meta directory has to be defined. This structure must be kept general, so that all kinds of directories can be a component of the meta directory. As second, it is shown, how one can access (read and write) data of the legacy system. As third a transformation shall be realized between the data of the legacy system and the meta directory. The main attention lies on describing the idea in principle and the proof of the concept with a prototype application.

This concept shall be carried out at the example of a university with students as business objects and SAP CM as legacy system. Therefore the meta directory consists of student, employee and guest information and has to be synchronized with the data of SAP CM. In order to proof the concept we will synchronize the student data of the two systems.

1.1.3 Processes

In this part of the document, we tackle on the modeling of the planned processes for the application.

(1) How will data be extracted?

(2) And how will it be stored?

(3) What would be the operations realized between extraction of data and its final storage?

Information 5: Main Concerns of Synchronization Process

In Information 5 we pose initial questions that have to be handled by the process modeling, and will later guide us to a better design of the manipulation of data. Extraction and storage of data are respectively the initial and last operations of the synchronization process. They are the end-of process operations. What comes between has to operate on data while respecting the input and output interfaces of the source (meta directory) and target (SAP CM) systems.

Now we should pose ourselves the following questions about the desired data flow operations:

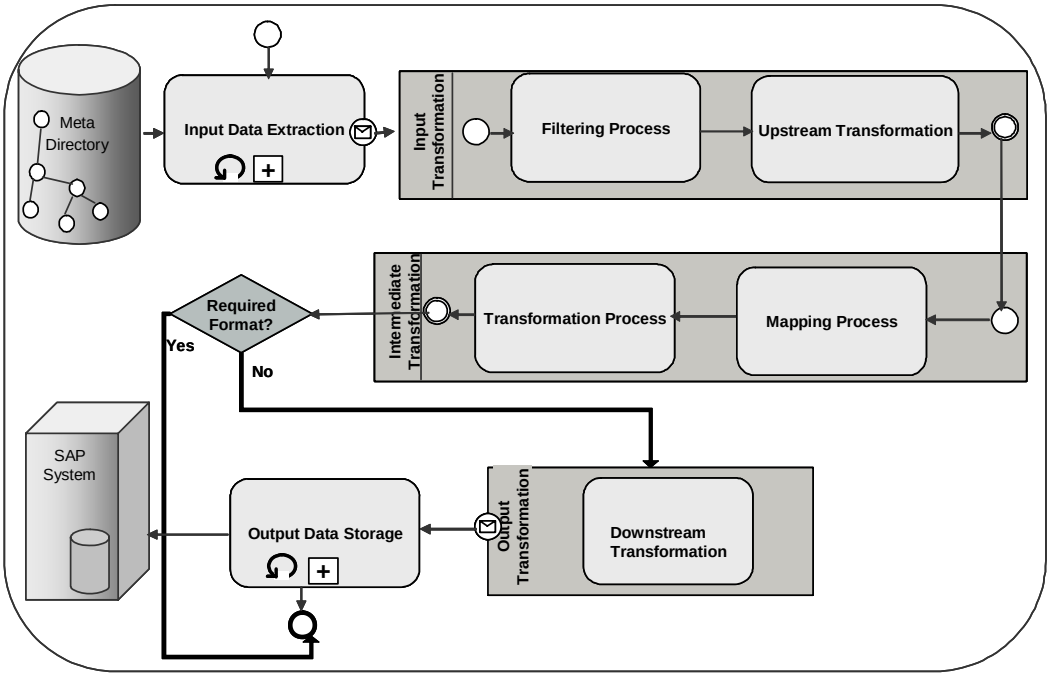
(1) How could these operations be organized in workflow?

(2) How could we organize these tasks in processes

(3) What information could be needed internally by the process?

Information 6: Process Modeling Decisions

Information 6 lists some questions related to decisional content of the processes and their organization. We have as well to define what information will be passed from one process to another. Answers to these questions make the logical separation of the processes clearer. The following process diagram proposes a simple answer to our questions.



Information 7: Design of the Synchronization Process

In Information 7 we present a global model of the synchronization process. In this view we distinguish three pools of processes and two separate processes. These latter are the “Output Data Storage” and the “Input Data Storage” Processes. Both are end-of-flow processes and have direct access to the manipulated persistent data. How this access is structured and realized will be explained in the next chapter.

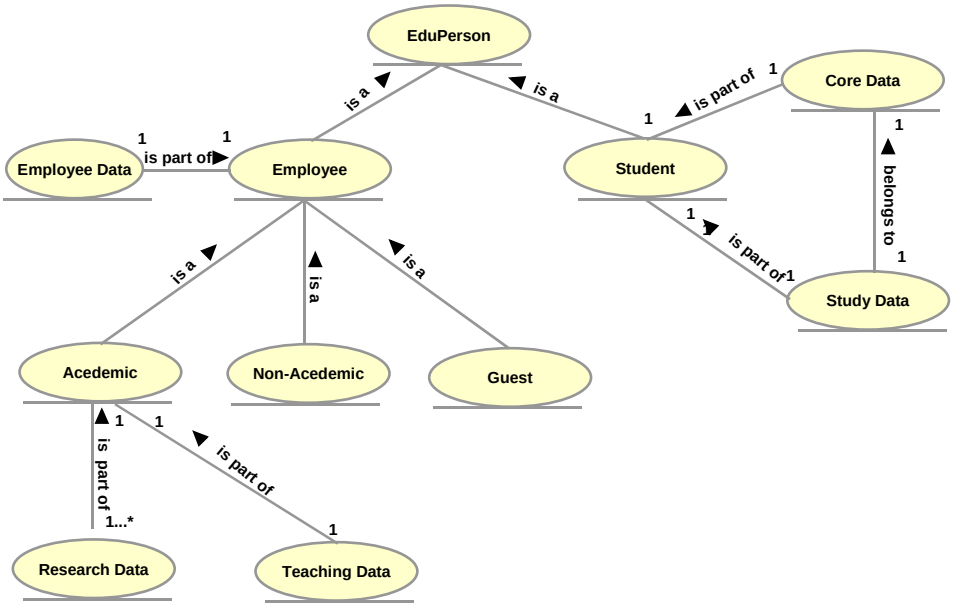
In the BPMN Model, the Data source is the meta directory and the data target is the URP system. In the “Input Transformation” pool, we have two sub-processes, the “Filtering” process and “Upstream Transformation” process, which are realized sequentially. The filtering process does some filtering on input data so that it can be better processed in later stages, such as elimination of irrelevant data or elimination of non synchronizing-exploitable data. The “Upstream Transformation” process is optional and leads some pre-matching transformations that may be needed before any mapping between the required data source and destination formats.

As a next step, in the “Intermediary Transformation” pool, we’ll have two other processes, The “Mapping Process and the “Transformation Process. The “Mapping Process” proceeds to the necessary correspondence translations between legitimate source and target formats. The “Transformation Process” is somewhat similar to the “Upstream Transformation Process” since

it also processes transformations on the data flow that are required by the synchronization and have to be fully specified for each instance of the synchronization. The two pools have been separated since the Mapping and Transformation processes have been modeled as compulsory, whereas the other processes in the two other pools are not always required and a synchronization transaction will be validated even if none of these processes has been executed.

Finally, an optional step in the workflow, the "Downstream Transformation Process", will be executed when there is still need for transformation before the output data can be written in the target repository.

1.1.4 Business Objects



Information 8: Business Object Diagram

The main focus of our case study is management and synchronization of student data objects between a legacy system and a directory service, the SAP CM system and the LDAP directory service But in a university directory service, where we manage data of different individuals involved within the university, we do not only store information about students. We also need to hold information on individuals like employees etc. Therefore, as business objects we consider generally all the individuals involved within a higher educational institute.

In an abstract view, we can generally identify all the people who are involved within a university as educational personal (EduPerson). They have common attributes like surname, first name and address. One can go one step further down in the abstraction, where you find the two main individuals in a university, students and employees. Although they have common attributes, with respect to business aspects they are different from each other in many ways. For example, employees have specific information on their contract with the university and their salaries and number of hours they work and on the other hand, students have information regarding their studies for example, current academic semester and information regarding their registration with the university, for example matriculation number and date of matriculation.

We can specialize employees further, because in a university or in a higher educational institute generally there is academic and non-academic staff. The academic staff is mainly involved in teaching and research in the university. The non-academic staff is mainly involved in administration operations in the university. An employee, who is a academic staff member, will

have information on his or her research projects and teaching modules and a non-academic staff member may have information regarding her or his administrative operations. In many universities there are guest professors, lectures and researches. They have come to a particular university to work for a shorter period of time, information regarding them should also be considered as business objects. Since their shorter period of stay at a university, they may have special information regarding their contract with a university.

In this section we have tried to give an introduction conceptually to the business objects that we must consider in designing our system. The meta directory should store information about all the individuals within a university we have discussed above. The SAP system, on the other hand basically offers the functionality to manage business process regarding students and their data. Although information about various individuals is stored in the directory and the legacy system, our main task is to synchronize student data between these two systems.

1.2 System Area

1.2.1 Meta Directory

(1) Specialized database optimized for reading, browsing and searching

(2) Data generally read more than written to
  (1) No transactions
  (2) No rollback

(3) May be local or global
  (1) Local : Directory services on a single machine
  (2) Global : Internet Domain Name System (DNS)

Information 9: Introduction to Directories

A directory is a specialized database optimized for reading, browsing and searching. Directories contain descriptive, attribute-based information and support advanced filtering capabilities. The main difference between database management systems and directories is, that directories generally do not support complicated transaction or roll-back schemes and directory updates are very simple all-or-nothing changes if they are allowed at all. On the other hand directories are tuned to give quick response to high-volume lookup or search operations.

There are many different ways to provide a directory service. A directory service may be local or global. In the first case they provide services to a restricted context and in the latter case to a much broader context, for example internet. Basically global services are distributed on many machines, but they provide a uniform namespace which gives a uniform view of the data you accessed. The Internet Domain Name System (DNS) is an example of a globally distributed directory service.

LDAP

In our system for accessing directory services we are using LDAP which is an acronym for Lightweight Directory Access Protocol. LDAP is a lightweight protocol for accessing directory services. This runs over TCP/IP or any connection oriented service.

The information model of LDAP is based on entries. An entry is a collection of attributes that has a globally unique Distinguished Name (DN). Using the DN we can refer to an entry uniquely. Each attribute in an entry has a type and one or more values. The types are like

strings, for example cn for common name, or mail for email address. The syntax of values is based on the attribute type, for example, a mail attribute of an entry would contain the value iswa.cm-tm.uka.de.

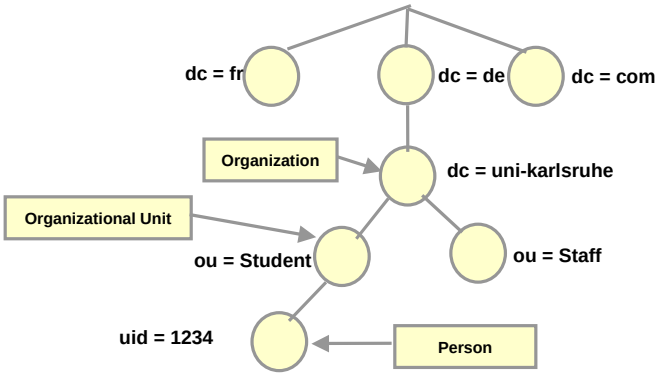
In LDAP the directory entries are arranged in a hierarchical tree-like structure. This tree-like structure reflects the organizational and geographic boundaries. Information 10 below shows an example LDAP directory tree using domain-based naming.

(1) LDAP = Leightweight Directory Access Protocol

(2) Runs over TCP/IP or any connection-oriented service

(3) Based on entries
  (1) Collection of attributes
  (2) Has a distinguished name (DN)

(4) Hierachical tree-like structure



Information 10: Introduction to LDAP

LDAP Schema

(1) Set of rules that describes what kind of data is stored

(2) Helps to maintain consistency and quality of data

(3) Object classes and attributes define schema rules

(4) A schema contains
  (1) Required attributes
  (2) Allowed attributes
  (3) What attributes can store

Information 11: LDAP Schema

A directory has a schema similar to the schema of a relational database system. The schema defines the content and the structure of the directory tree [S-SCHEMADES]. In an LDAP directory, the schema is the collection of defined attributes and defined object classes to control where which data is stored. As we have mentioned above an entry in a LDAP directory is a collection of attributes and all the attributes which may be used for a specific type of object are called Object Classes. A LDAP schema defines attributes and object classes giving you the

possibility to create new value types that may be stored in the directory and new object classes to meet your needs. Within each object class you may designate that some attributes are required, and that others are optional. You can also define and restrict information that can be stored in attributes. In traditional database point of view you can think fields are similar to attributes, tables are similar to object classes. Next we will explore the two concepts attributes and Object classes in a LDAP schema.

Attributes

(1) Attributes have :
  (1) Name: unique identifier
  (2) Description: definition and intended usage
  (3) Object identifier (OID) : globally unique sequence of integers
  (4) Attribute syntax
    (1) Data attributes can store – eg integer, string etc
    (2) How comparisons are made
  (5) Multi –or single valued ?

Information 12: Attributes

An attribute is a container that may be used to store a single type of information within the directory. For example a student entry may have a matriculation number and a first name each of them describes an individual attribute of the student. The schema allows to designate as many types of attributes as needed. When defining an attribute the following aspects must be specified.

- The attribute's name: Although this is a straightforward concept, there are some guidelines defining them. The name must not conflict with a standard attribute that is already defined, therefore the best practice is to define the name beginning with the name of the organization, for example "unikarlsruheFirstName".
- Description: A one-sentence definition of the attribute's intended usage
- Object Identifiers: Every attribute type is identified by globally unique string of integers (the OID).
- Syntax used for type checking and for pattern matching
- Whether or not more than one value of this attribute is allowed per entry. *(Implementation dependent)*

ObjectClasses

(1) Used to group information

(2) Define following rules
  (1) Required attributes
  (2) Allowed attributes

(3) Object classes can be derived from others

(4) Can extend attributes of other object classes

(5) Entries can have multiple object classes

(6) Three types of object classes
  (1) Structural
  (2) Auxiliary
  (3) Abstract

(7) Object classes define:
  (1) Name, OID, Description and Super class

Information 13: Object Classes

In LDAP each entry belongs to object classes that identify the type of the data represented by the entry. Basically an object class specifies the mandatory and optional attributes that can be associated with an entry of that class. The object classes for all objects in the directory form a class hierarchy. The classes top and alias are at the root of the hierarchy. For example, the eduPerson object class is a subclass of the Person object class, which in turn is a subclass of `top`. When creating a new LDAP entry, all of the object classes to which the new entry belongs, should be specified. There are three types of object classes:

- Structural: Indicates the attributes that the entry may have and where each entry may occur.
- Auxiliary: Indicates the attributes that the entry may have.
- Abstract: Indicates a partial specification in the object class hierarchy; only structural and auxiliary subclasses may appear as entries in the directory.

When defining an object class we must specify the following

- OID: Unique object identifier
- Name: Object class's name
- Description: Object class's description
- Obsolete: "true " if obsolete, "false" or absent otherwise
- SUP: Names of superior object classes from which this object is derived
- Abstract: "true" if object class is abstract, "false" or absent otherwise
- Structural: "true" if object class is structural, "false" or absent otherwise
- Auxiliary: "true" if object class is auxiliary, "false" or absent otherwise
- Must: List of type names of attributes that must be present
- May: List of type name of attributes that may be present

As we have discussed the basics of LDAP and LDAP schema design, now we can introduce the schema design aspects for our directory service.

eduPerson Schema

(1) Auxiliary object class for campus directories

(2) Facilitates interoperation in higher education

(3) Sponsored by EDUCAUSE and Internet2

(4) Defines attributes about individuals in higher education

(5) Supports inter-realm applications
   (1) Controlled access to web pages or licensed resources
   (2) Allowed attributes
   (3) Example : Secure resource sharing of scientific research among universities.

(6) Must be used with other person schemata (person, inetOrgPerson etc)

Information 14: eduPerson Schema

eduPerson is an auxiliary object class for campus directories designed to facilitate communication among higher education institutions. It consists of attributes, about individuals within higher education, along with recommendations on the syntax and semantics of the data that may be assigned to those attributes. It is said that, if widespread agreement and implementation of this object class in campus directories is achieved, a broad and powerful new class of higher education applications can be deployed. For additional information refer to [EDUPERSON01].

The key feature of eduPerson attributes are that they are intended to support inter-realm applications such as controlled access to web pages or licensed resources. Many of these inter-realm applications are just beginning to appear, for example Directory of Directories of Higher Educational institutes. Most of them are related to instructional and research use, for example, Web pages associated with a course at one university could be easily and securely opened to students in another course in another university and scientific researchers could reserve specialized computing resources at distance locations using local services.

Integrating eduPerson schema to your own directory the other person schemas must be used like person, organizationalPerson and inetOrgPerson. Because other attributes that describe a person generally and a person in an organization are in them. The eduPerson schema concentrates only on specialized aspects of a person in the higher education and the other relevant attributes must be used from the other schemas.

Attributes in eduPerson

(1) eduPerson defines following attributes
  (1) eduPersonAffliation
  (2) eduPersonPrimaryAffliation
  (3) eduPersonOrgDn
  (4) eduPersonOrgUnitDn
  (5) eduPersonPrincipleName
  (6) eduPersonNickName
  (7) eduPersonEntitlement
  (8) eduPersonPrimaryOrgUnitDn
  (9) eduPersonScopedAffliation

(2) All these attributes are optional

Information 15: Attributes in eduPerson

In eduPerson schema all the attributes are prefaced with eduPerson and all the attributes are optional, i.e. an eduPerson auxiliary object class contains all of the attributes as type MAY. Now a brief introduction to the attributes in eduPerson object class is given.

LDAP ATTRIBUTE NAME	ATTRIBUTE CHARACTERISTIC	ATTRIBUTE DEFINITION	ATTRIBUTE VALUES
eduPersonAffliation	Multi Value	Specifies the persons relationship(s) to the institutions in broad categories.	Examples: student, faculty, employee
eduPersonPrimaryAffl-iation	Single Value	Specifies the person's primary relationship to the institution in broad categories	Examples: student
eduPersonOrgDn	Single Value	The distinguished name (DN) of the directory entry representing the institution with which the person is associated	Example: o=informatik, de=uni-karlsruhe, de=de
eduPersonOrgUnitDn	Multi Value	The distinguished name(s) of the directory entries representing person's Organizational Units	Example: ou=student, o=informatik, de=uni-karlsruhe, de=de
eduPersonPrincipleN-ame	Single Value	The "NetID" of the person for the purposes of inter-institutional authentication	Example: person@uni-karlsruhe.de
eduPersonNickName	Multi Value	Person's nickname or informal name	Example: Joe , Cath
eduPersonEntitlement	Multi Value	URI that indicates a	Example: http://cm-

		set of rights to specific resources	tm.uka.de/contracts/ Perera
eduPersonPrimaryOr-gUnitDN	Single Value	The distinguished name (DN) of the directory entry representing the person's primary Organizational Units(s)	Example: ou=student, o=informatik, de=uni-karlsruhe, de=de
EduPersonScopedAff-Iation	Multi Value	Specifies person's affiliation within a particular security domain	Example: faculty@informatik.uni-karlsruhe.de

Many institutes in USA have already agreed to implement the eduPerson object class and some other institutes have said that they will do so when they implement an enterprise-wide directory. In contrast some of the universities in UK have adopted the eduPerson schema by extending some of its features for the social and operational requirements. Currently we could not acquire precise information on institutes in Germany who have adopted this schema or used it for their educational directories. In the next section we would like to introduce how we can use this schema or adopt this schema to represent the meta directories in our system.

Analysis: "Is eduPerson sufficient?"

In order to use eduPerson schema to model student entries in our directory, we would definitely like to perform a broader and a complete analysis of eduPerson by taking on discussions, interviews, questionnaires with people who are involved in higher education and those who develop software systems for academic work. But the time we have for our case study does not allow us to go in such broader analysis. Therefore we have used and gained knowledge from an analysis done by the company DAASI International on one of their projects (DEEP) Definition of a European EduPerson based on a questionnaire [DEEP-A2]. Next we will present a short description of the results of the analysis that are relevant to us. For more information please refer to [DEEP-A2].

(1) eduPerson is not sufficient

(2) Majority of attributes in eduPerson are relevant

(3) Suggested enhancements to eduPerson
  (1) Identity
    (1) SocialSecurityNumber, unique_userid, studentnumber
  (2) Information
    (1) personTitle, position
    (2) areaOfInterest, expertise, studyBranch
    (3) birthDate, cv, gender

Information 16: Overview of the DEEP Questionnaire

For the question "Do you think the current eduPerson definitions are sufficient for the European research community?" only 11% of the participants saw eduPerson as sufficient and 38% of the participants saw it insufficient and the rest no opinion. As we have already mentioned this case study has also analyzed not only the eduPerson but the whole person schemas generally because eduPerson in isolation cannot model a solution in modeling a person in higher education.

The results for the relevancy of attributes in all 4 schemas can be summarized as follows for further details please refer to the literature stated above, because our discussion will continue concentrating on eduPerson

- All attributes of object class person
- 8 of the 17 attributes of the object class organizationalPerson
- 18 of the 28 attributes of inetOrgPerson
- 7 of the 8 attributes of eduPerson

Although majority of the attributes of eduPerson are seen as relevant, but as we have seen above, the schema itself is not sufficient to model people in higher educational institutes. Also they have concluded that the attributes eduPersonOrgDN, eduPersonPrimaryAffiliation, eduPersonOrgUnitDN, eduPersonAffiliation have been seen as highly relevant attributes.

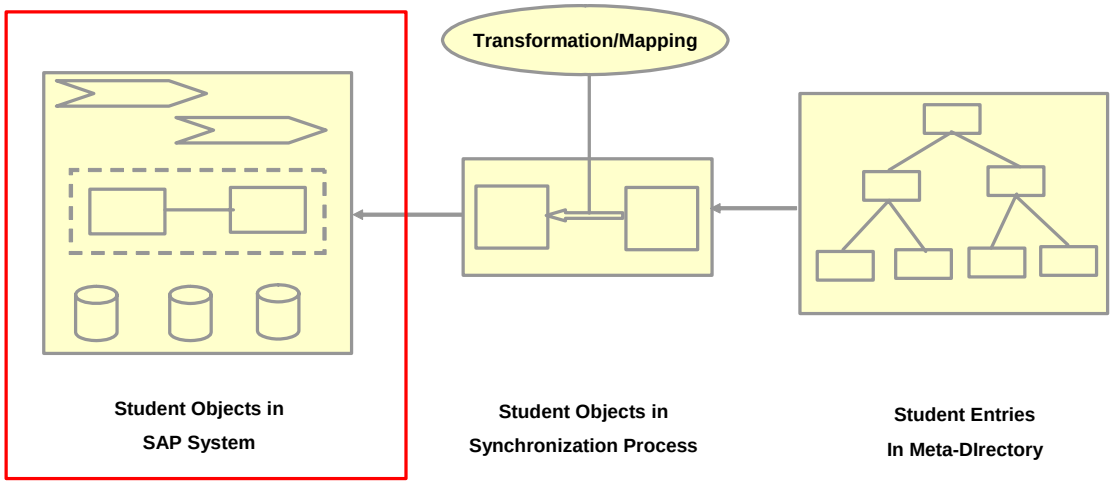
In the next phase the participants answered the question "Which additional attributes would you need?". There were 20 different attributes and attributes relating to a persons information and identity are as follows

1. Identity
   - SocialSecurityNumber, unique_userid, studentnumber, fedID, netID
2. Information
   - personalTitle, position
   - areaOfInterest (together with classificationScheme), expertise, studyBrancg
   - birthDate, cv, gender

The conclusion of this analysis in our view is eduPerson schema has relevant attributes but it is not sufficient enough to use it in higher educational institutes in countries in Europe. One of the main reasons is that, very important features in the European context like student number or what we use in most of the German universities the matriculation number and birth date are missing in eduPerson.

1.2.2 Legacy System

Legacy Systems in General



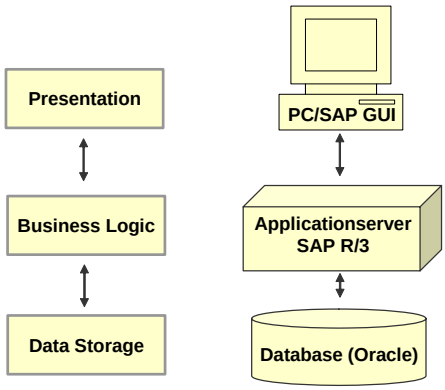
Information 17: Location of the Legacy System in our Scenario

A legacy system is an existing software solution that has been used for a very long time, because an enterprise doesn't want to replace it. The reasons for this behaviour are different. For example they are very large, complex and monolithic, so that it's very complex to buy a similar system with the same functionality and it is. Some of them require close to 100% availability, so that it can't be taken out of service. And sometimes the system works satisfactory, so that the owner doesn't see any reason to replace it. Mostly, they are problematic, because it is very difficult to maintain, improve or expand them. The company, who developed the system has the fundamental problem to understand and adapt the software. After some time there is a lack of knowledge about the software. It's because the designers left the organisation and inadequate documentation gets lost over the time.

On the other hand it is important to mention, that some of the legacy systems are very successful over a long period of time. The developers had very much time to optimize and adapt the software in a way that it exactly meets the demands of the users. The users know the software and have a very good feeling of the advantages and disadvantages of the software. So the fact that legacy systems are very old and still in use can also be seen in a positive way. Although from the technical point of view it is designed and developed by very old methods and programming languages, it is in use for many years. Because of this the users doesn't have to learn a new system and they don't have to be coached for using the new software. The replacement of an old and productive legacy system has a lot of drawbacks. For example is the first version of a new software not working properly as well. An even with new programming techniques and the use of a modular and service oriented architecture, the effort for developing a similar system with the same functionality is also very high. And it can't be expected that the first version of a newly software will have the same functionality and availability of an established and for many years used legacy system.

In this case study, we are primarily interested in the student data of SAP CM. Therefore, we will first give a summary of the architecture of the legacy system SAP R/3. Then we briefly explain the functionality of SAP CM and describe more exactly how the students are represented in SAP CM. Here, it is important which data of the students is needed exactly, to be able to carry out the synchronization successfully.

Architecture of SAP



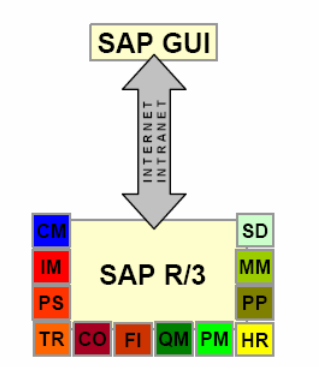
Information 18: 3-Level Architecture of SAP R/3

SAP uses a client server architecture at their system. The system is divided in three layers. The presentation, the business process and the database layer. TCP/IP is used for the transport. The presentation is the interface to the user and is realized for all applications with a windows client, the so-called SAP GUI. In this scenario, the business process logic as well as the database layer is put down physically on the same server. The logic forms the interface between the presentation-

and the database-layer. It takes on the requirements of the user from the presentation layer and executes the instructions and evaluations. The data necessary to this is requested by the database over an SQL interface. SAP is software which is written in a development environment of its own. The SAP programming language ABAP/4 is a union of COBOL and SQL. The SAP R/3 system component forms the server-core-component of the architecture mentioned above to which several optional modules can be docked.

In the context of our WUSKAR Project the SAP R/3 system is operated by the computer center of the Universität Karlsruhe (TH).

SAP Campus Management

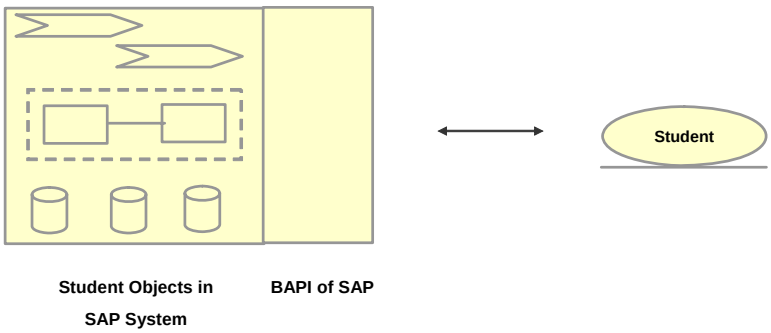


- Campus Management (CM)
- Sale (SD)
- Materials management (MM)
- Production planning (PP)
- Quality management (QM)
- Maintenance (PM)
- Human Resources (HR)
- Financial system (FI)
- Controlling (CO)
- Treasury (TR)
- Project system (PS)
- Investment management (IM)

Information 19: Different SAP modules

SAP offers a variety of modules. These are the heart of the system from a manager's viewpoint. These modules may not all be implemented in a typical company but they are all related and are listed above. One of these modules is SAP CM. SAP CM fulfils a variety of needs of the universities. For example it offers the opportunity to manage students and employees of the institution.

BAPI

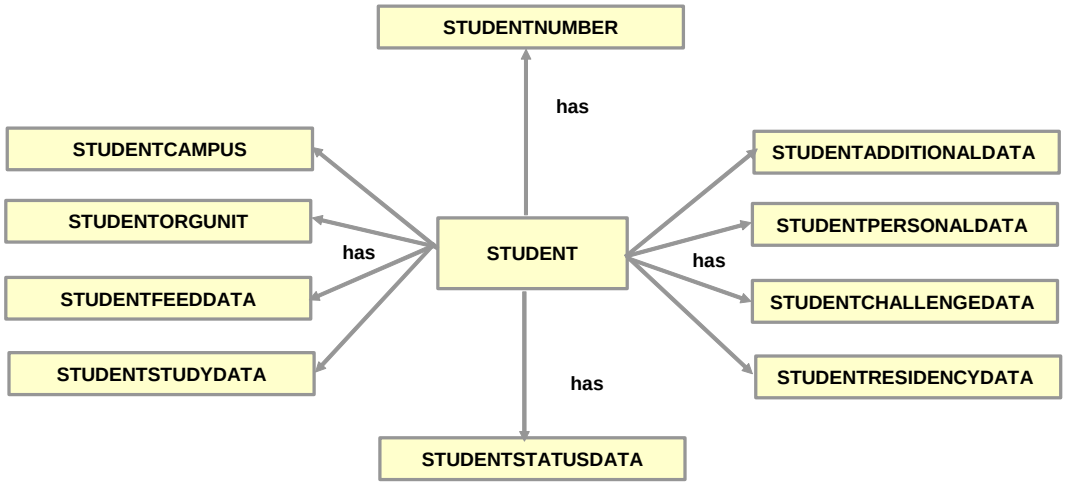


Information 20: Access to the Business Object Student over BAPI

In order to offer an interface on higher abstraction level, SAP developed the Business Framework. This framework offers an object oriented structure of the exchanged data between the SAP system and the application, which accesses the framework. An integral component of the framework is the so-called (BAPI) business application programmer interface. The BAPI represents an object-oriented representation of business objects and put the application developer of an SAP access component into the situation to import or export the demanded business objects.

In the context of our synchronization process we confine ourselves to the business object student. These business objects are also represented in SAP CM. SAP CM makes therefore different BAPIs available, with which one can access these student data. In the context of our example the object student is imported through the synchronization process from the meta directory over the BAPI into the legacy system. The BAPI makes the import of the student into the SAP CM possible. Here the BAPI consequently makes the combination between the business object student and the internal representation of the student in SAP CM possible.

BAPI Student Object

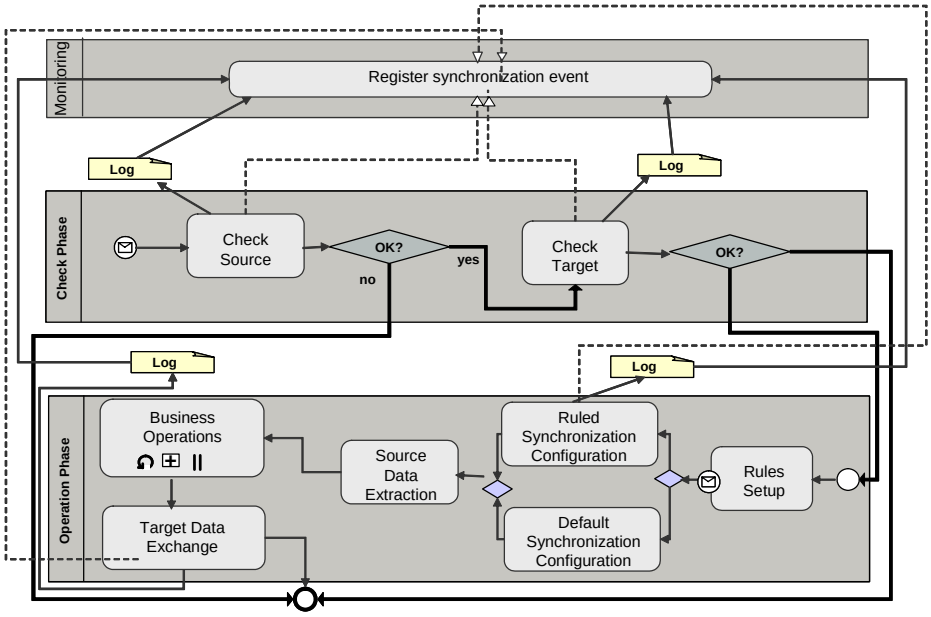


Information 21: Overview of the Student Data Structure of the BAPI

In order to import the data from the meta directory into the SAP system, we use the BAPI "BAPI_STUDENT_CREATEFORMDATA3". This BAPI is provided by SAP and described as the BAPI, which has to be used for importing student objects. This BAPI offers all semantically relevant information and attributes, which are assigned to a student. Through this, one can deduce by which attributes a student is represented in SAP CM. In the following table depicts an abstract description, which kind of information is assigned to a student at the BAPI.

Attribute	Description
STUDENTNUMBEREXTERN	Registration number of the student
Table	**Description**
STUDENTPERSONALDATA	Personal data of the student
STUDENTADDITIONALDATA	Additional data of the student
STUDENTCHALLENGEDATA	Handicap of the student
STUDENTRESIDENCYDATA	Residence data of the student
STUDENTSTUDYDATA	Studies data of the student
STUDENTFEEDATA	Charges relevant data of the student
STUDENTORGUNIT	unity, to which the student is assigned to (faculty)
STUDENTCAMPUS	location, the student is assigned to
STUDENTSTATUSDATA	Status attribute of the student

1.2.3 System Process



Information 22: Ruled Synchronization & Event Registration

In Information 22: Ruled Synchronization & Event Registration we present another view of the elaborated business process. There are three pools. The "Monitoring" pool contains the events monitoring utilities. Their task is to catch events and messages that are sent from the system sub-processes and make them available at the interaction layer. For a better understanding of this task, see the integration layers diagram in the design phase. The "Register synchronization event" catches the "Log" messages.

(1) In the "Check Phase" pool, we have a set of pre-operational processes. With initial information about the data source repository, the "Check Source" performs an access test to this source. If it is unsuccessful it goes directly to the end-event. It sends an information "Log" message to the "Register synchronization event" process. If it is successful, it performs a target data repository access test. Again, if the test is unsuccessful, then the workflow is redirected to the end-of-process procedure. Else, the initial event in the "Operation pool" is started.

(2) In this pool, the first process takes in charge the gathering of rules, if any. The system must be able to work when there are no rules defined, with default behavior. Rules are a set of parameters that can parameter the behavior of the synchronization process. These rules are not compulsory, and will ideally be defined completely separately from the core data flow processing. More explanations about the rules system will be given in the "Synchronization" paragraph of the design phase.

(3) Then, provided the set of rules is gathered the synchronization core processes are configured, either with rules or with default configuration. The synchronization core processes are those presented in the first view of the business process model. They will be presented with more details in the "Synchronization" part of the design phase.

(4) After the configuration of the core synchronization processes is done, source data is extracted, then processed, then stored in the target repository. The "Source Data

Extraction" and the "Target Data Storage" processes use some predefined interfaces which description you'll find in the synchronization part of the design phase.

(5) The "Business Operations" process is the process that executes the core synchronization workflow. It is an iterative and complex (means composed of sub-processes) process. Notice that after the synchronization configuration, the "Source Data Extraction" and the "Target Data Storage" messages are sent to the "Register synchronization event" process.

We have not taken security concerns into consideration in this process modeling yet. Nevertheless they are part of the design. They may be realized in additional development increments (an increment is a complete realization of a set of chosen system functionalities that can be standalone delivered, and can be seen as implementation cycles of an application). At this stage, the synchronization rules will have to be completed and we may need additional monitoring or utility security processes.

1.3 Exercises

(1) Why do we use a centralized meta directory ?

(2) What operations will be led between the extraction of data and its storage?

(3) What mechanisms enable the management of several contexts for the synchronization process?

(4) Describe the difference between a directory and a database.

(5) What is defined by a schema in LDAP ?

(6) Why does the used BAPI combine the IT area with the business area ?

Information 23: Exercises

(1) Information 4: Scenario of the Synchronization Process

(2) Information 7: Design of the Synchronization Process

(3) Information 22: Ruled Synchronization & Event Registration

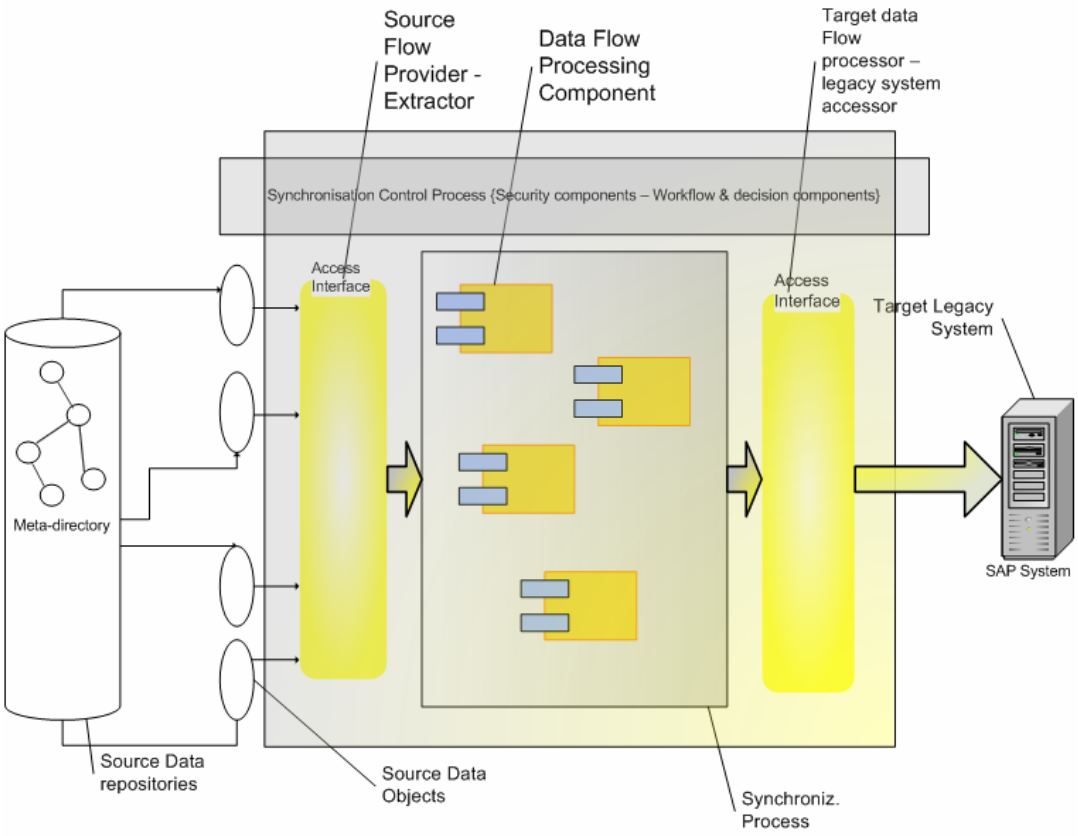
(4) Information 9: Introduction to Directories

(5) Information 11: LDAP Schema

(6)

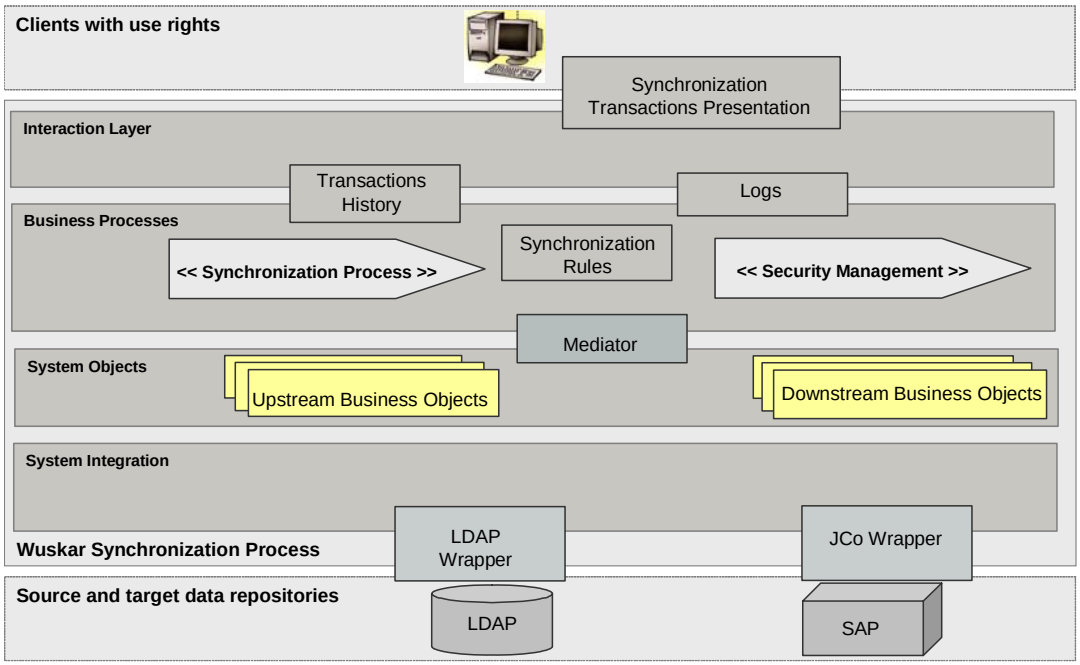
2 DESIGN

2.1 Application Architecture



Information 24: Global Application Architecture

In Information 24 we give a complete view of the application architecture through a system model. The label “Synchronization Control Process” relates to the upper layer control process that provides workflow decision and security support to the “Synchronization Process” (see Information 24: Global Application Architecture). The initial components are “Source Data Objects” extracted from the “Source Data Repository”.
Through an “Access Interface” they will be passed to a set of components that form the “Core Synchronization Process” primitives. These primitives are called “Data Flow Processing Components”. A later stage is to go through another “Access Interface” into the Legacy System. Note that here both architecture elements interacting with the end systems are both called “Access Interface” because they are actually managed by the same components, as we’ll see in the synchronization subsection later in this section. How these “Data Flow Processing Components” are constructed and how they collaborate together is explained later in this design section.



Information 25: Integration Layers Diagram

Information 25 gives a Layers diagram of the application. Several layers have been distinguished. First, the "Interaction" layer" is the layer that takes in charge communication with the application layer where it can for example display information about the synchronization process and the ongoing operations. It can for example as well manage many configurations for the synchronization or offer a GUI to build up a synchronization instance. Possible services may be displaying a list of the available meta-directories according to the rights of the current user of the application.

In the "Business Processes" layer will be located such processes as the "Core Synchronization Process" itself or possibly a security management process. Two system components interfaced between the "Interaction" layer and the "Business Processes" layer are the "Logs" component, which stores all messages coming from underlying components and destined for the Interaction layer. As well as the "Transactions History", which stores information about the currently led operations, which can be of use as a GUI view for the synchronization process administrator for example.

One layer below we find the "System Objects" layer. It simply contains the transactional data flow objects, which are to be processed by the overlying "Core Synchronization Process" (CSP). Two types of instances are distinguished, the "Upstream Business Objects" and the "Downstream Business Objects". "Upstream Business Objects" are simply the source data objects and respectively, "Downstream Business Objects" are target data object, which are the result of the operations realized by the CSP. The mediator component is the entity that manages these business objects, in the way that it takes care of their extraction and storage logics as well as providing them to the CSP.

In the "System Integration" layer we have the two following components, the "LDAP Wrapper" and the "JCo Wrapper" which are also interfaced with the base layer of "Data Repositories". They both respectively manage access and dialogue with end-of-system collaborators, namely the meta-directory server and the URP system.

2.2 Interface of the Directory

In this section we present a conceptual description of the interface of our Directory. Since our system deals with directory synchronization, we need a clearly defined interface to the student data in the meta directory, which we access through LDAP, so that the synchronization process or an application has a consistent interface to the data and it can easily access the student data objects in the directory without considering the underlying implementation of the Directory Information Tree (DIT).

First we will introduce the schema design for the LDAP, which ensures that applications have a consistent interface to the data. Next we will describe the communication between the synchronization application or any application with the LDAP server and finally we will describe, how an application communicates with the directory through the interface.

2.2.1 LDAP Schema Design

(1) Ensure applications have a consistent interface to data

(2) Maintain consistency and quality of entries in the directory

(3) Use of the existing schema for campus directories eduPerson

(4) Schema design proposals, model schema with,
   (1) organizationalPerson, inetOrgPerson and eduPerson
   (2) organizationalPerson, inetOrgPerson and our own schema deEduPerson
   (3) organizationalPerson, inetOrgPerson, our own schema kaEduPerson and eduPerson

(5) Designing our own schema deEduPerson

Information 26: Schema Design

To ensure that applications have a consistent interface to data and to maintain consistency and quality of entries in the directory we need to define a schema for LDAP. The background and an introduction to LDAP schemata and an analysis of an existing schema eduPerson, which defines a schema for campus directories designed to facilitate communication among higher education institutions are given in Chapter 1.2.1. The main task at this point is to design and specify a schema to model information of individuals in our meta directory.

As we have already mentioned in Chapter 1.2.1, although some educational institutes, mostly in USA, are using the eduPerson schema, the European institutes express that it is not sufficient to represent persons in European institutes. In our opinion, the main reason for this may be the differences in social and operational requirements in USA and in Europe.

In order to develop a schema for the representation of student objects in our directory we have analyzed 3 proposals stated in Information 26.

We have discarded the first proposal because we have already seen in analysis that eduPerson only is not sufficient enough. The main idea behind the second proposal is that we will discard the eduPerson schema and develop our own schema to represent students. Though, this sounds a good suggestion we loose some of the important attributes in eduPerson like eduPersonOrgDn and we are going away from the international standards and we would not be able to benefit from a standard schema that supports communication among higher educational institutes

internationally. So we have decided to stick to the last option, where we keep eduPerson schema and develop our own schema edeEduPerson extending and using the other person schemas.

First we will describe the attributes that we have introduced in our schema deEduPerson and next we will give a brief introduction of the attributes that we have taken from the other person schemas including eduPerson, which gives us a vast flexibility especially in representing student data in our meta directory.

deEduPerson Schema

(1) deEduPerson defines following attributes
  (1) UserId (uid)
  (2) Matriculation Number
  (3) Field of Study
  (4) Date of Matriculation
  (5) Academic Semester
  (6) Gender
  (7) Date of Birth
  (8) Nationality
  (9) Person Title

(2) Only UserId is mandatory and all the other attributes are optional

(3) For students, UserId = Matriculation Number

Information 27: Attributes in deEduPerson

Information 27 shows the attributes that we have defined in our deEduPerson schema. In this schema other than UserId all the other attributes are defined as optional. Therefore, when implementing this schema, the attribute UserId should be defined. Now we will describe each attribute in detail.

- UserId (uid)
  One main drawback in eduPerson is, it does not define a unique id, which can identify a person in an educational organization uniquely. We have first thought of having the matriculation number, which we present next as the unique identifier, since it identifies uniquely students in higher educational institutes in Germany. On the other our directory should also hold information about professors, staff members, research assistance and administrative personal, therefore, we cannot solely use matriculation number as the unique identifier and we have defined UserId as the unique identifier generally to identify all the personal including students uniquely. Therefore, this attribute is defined as mandatory.

- Matriculation Number
  A matriculation number uniquely identifies a student who is matriculated at a university. This is a very important attribute related to European or German higher educational institutes because every student is identified uniquely in these institutes using her or his matriculation number. This is defined as an optional attribute, since we have the UserId, which is used for unique identification. The student entries will have the same value for the UserId as well the matriculation number.

- Field of Study

This attribute identifies the main field of study and was missing in eduPerson schema, especially in universities in Germany students are organized according to their main field of study.

- Date of Matriculation
  The date of matriculation defines the date in which the student is matriculated at the university. In the administration point of view this is very important attribute that tracks the whole history of the student stay at a certain university.

- Academic Semester
  Academic semester identifies the current study semester of the student. This attribute was missing in eduPerson. This attribute is required in German universities because they have laws and rules governing the number of semesters that a student can study in a particular university.

- Gender
  This was one of the main attributes that was missing in eduPerson or in any person schema. This attribute is very helpful in order to make statistical analysis of students who are studying in a particular university or universities in a specific country.

- Date of Birth
  In the analysis we have taken about eduPerson most of the people have suggested date of birth as an attribute that describes a persons information. Therefore we have decided to define it in our schema.

- Nationality
  We have introduced this attribute to our schema especially considering the number of international students who are studying in German universities. It is a very important attribute to track person data of student in a particular university.

- Person Title
  We have taken this attribute considering the suggestion given in the analysis we have taken into consideration. It does also play an important role, when we extend or use our schema to represent professors or staff members in an institute because they may have different personal titles, for example “Professor, Doctor, diplom-Ing”.

Attributes Taken from Person Schemata Including eduPerson

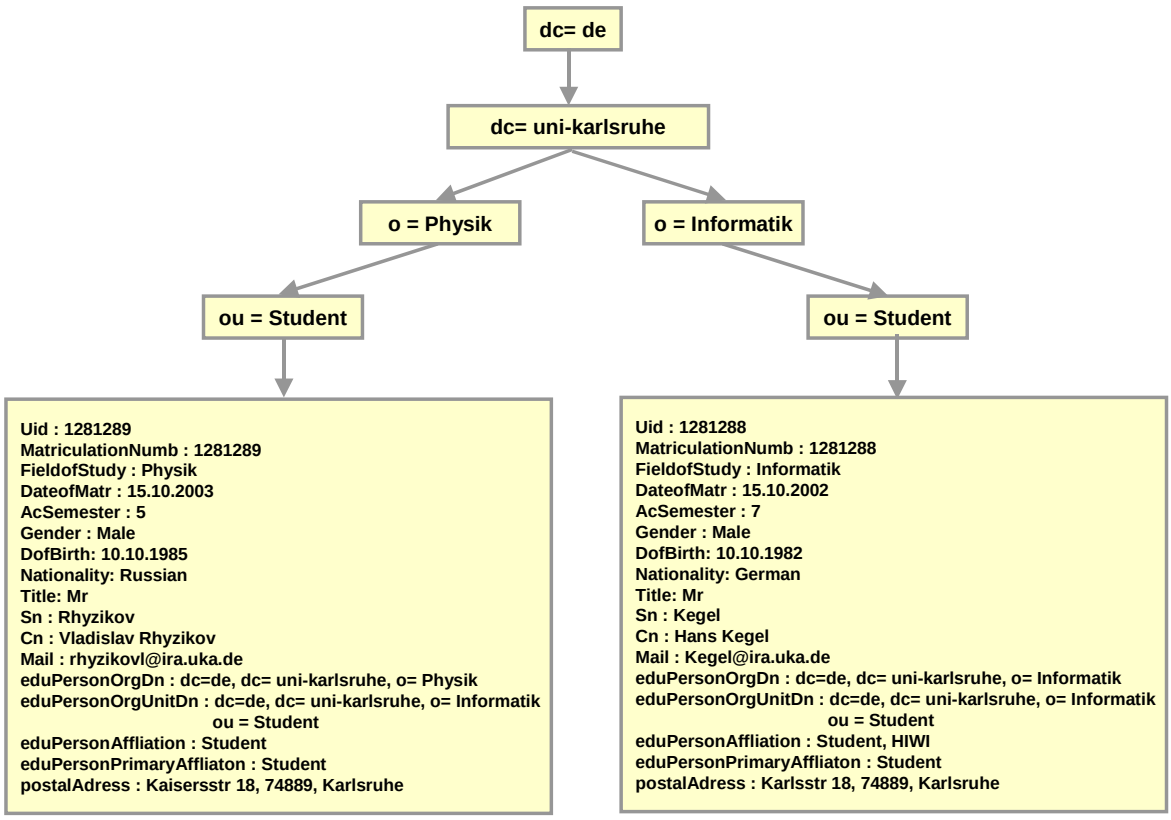
Common attributes that define a person, for example first name, surname and email address, we have taken from the person schemas and we also use the eduPerson schema which gives us the possibility and flexibility to define some of the important attributes in eduPerson and in the same time give us the possibility to confirm to international standards. The table below describes briefly those attributes we have taken from the person schemas.

Attribute	Object Class	Description
Sn	person	Family name of the person
Cn	person	Common full name of the person
homePostalAddress	inetOrgPerson	Home postal adress
homePhone	inetOrgPerson	Home telephone
postalAdress	orgPerson	Current postal adress
telephoneNumber	person	Current telephone number
Mail	inetOrgPerson	Preferred email adress
postalCode	orgPerson	Postal Code of current adress

Street	orgPerson	Street of current adress
userPassword	person	Identifies entrys password
givenName	inetOrgPerson	Name which is not the surname nor the middle name
eduPersonOrgDn	eduPerson	Distinguished name of the directory entry representing the institute with which the person is associated
eduPersonOrgUnitDN	eduPerson	The distinguished names of the directory entries representing the persons organizational units
eduPersonAffliation	eduPerson	Persons relationship to the university in broad categories
eduPersonPrimaryAffl -iation	eduPerson	Persons primary relationship to the university

Student Objects in the Directory

As we have defined the deEduPerson schema to represent student entries in our meta directory, we now present the valid Directory Information Tree (DIT) of student entries in the LDAP server.



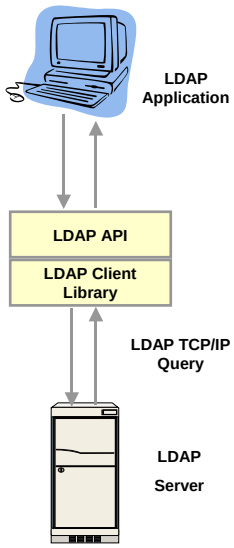
Information 28: DIT for Student Objects in the Directory

The DIT is diagrammatically represented in Information 28 above showing the naming of entries and their object classes. The top node is de, with the unique DN dc=de, where dc specifies a domain name. The first child node defines a sub domain uni-karlsruhe under the top domain de with the unique DN as dc= uni-karlsruhe, dc=de. The university will have different

faculties like Informatik, Economics and Physics. We identify these institutes as organizations in our DIT, therefore, the parent node dc=uni-karlsruhe will have two child nodes in this example o=Informatik and o=Physik. Each of these departments can be uniquely identified as o=Informatik, dc=uni-karlsruhe, dc=de and o=Physik, dc=uni-karlsruhe, dc=de. Each of these faculties will have organizational units, like students and staff. The parent nodes o=Informatik and o=Physik each will have a child node ou=Student. This defines a person's organizational unit, which he is assigned to. A staff member at the university will have the value ou=Staff. The actual student entries are under this node ou=Organizational Unit. Each child node of the parent node ou specifies an entry for a student, which is of structural object class "deEduPerson" we have specified earlier (see Chapter 1.2.1). Each student entry can be uniquely identified by the DN uid=xxxxxxxxx, ou=AA, o=BB, dc=uni-karlsruhe,dc=de. For example the distinguished name (DN) for a student with the uid=12812888 who studies Informatik can be identified by the DN uid=12812888, ou=Student, o=Informatik, dc=uni-karlsruhe, dc=de.
The DIT of the LDAP gives client application, in our case the synchronization process the possibility to access a student entry with its distinguished name (DN). Using this DN the application can modify, update or remove entries in the LDAP DIT.

In the next section we will introduce how an application can generally communicate with a LDAP server and the way in which an application can retrieve student data using a DN.

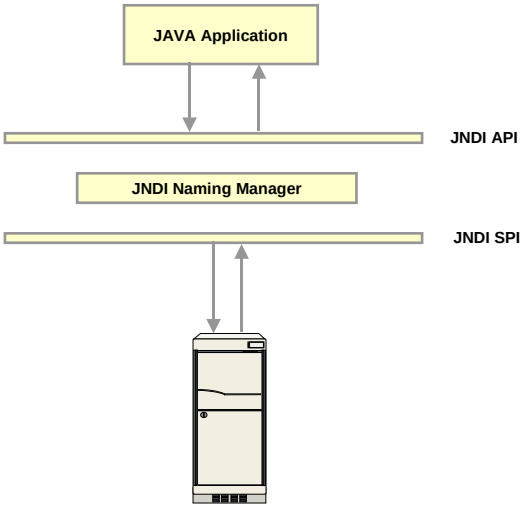
2.2.2 LDAP Communication



Information 29: LDAP Communication with an Application

Information 29 illustrates the communication of an application with the LDAP server. For this purpose a LDAP client library, which implements LDAP server specific code is required. This module should implement code specific to different LDAP providers. It has provider specific implementation which supports retrieving, modifying, and removing data stored in the directory. The communication between the client library and the LDAP server is realized using TCP/IP, therefore the client application can be deployed on a remote machine. Above the LDAP client library you have the LDAP API and any application that wants to connect to the LDAP server can call generic methods in the LDAP API without considering the implementation specific details. This layered architecture provides a high level logical view that hides physical details to any application communicating with the LDAP server and provides the application programmer the flexibility to concentrate only on logical and business aspects without considering the physical implementations. Since our synchronization process is developed in

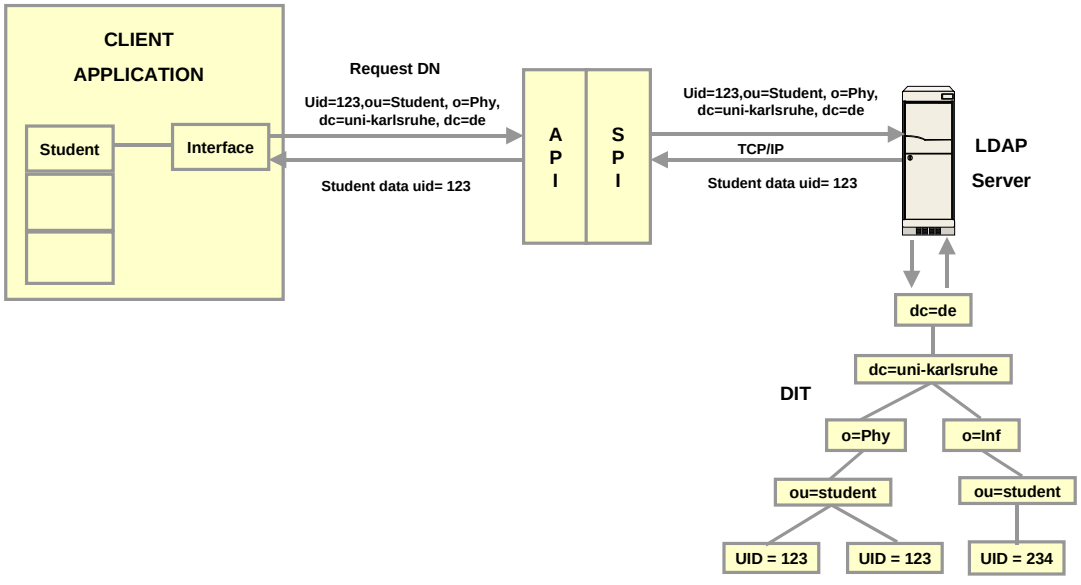
JAVA programming language, next we will describe the communication between a JAVA application and a LDAP server.



Information 30: LDAP Communication with a Java Application

A Java application has the possibility to access the LDAP directory service using JNDI (Java Naming and Directory Interface). JNDI is an API that supports accessing naming and directory services in Java programs. The architecture of the JNDI provides a standard protocol-independent API built on top of protocol-specific or provider implementations. JNDI provides an API to Java applications to access LDAP directory services. It provides a service provider interface (SPI), in order to actually interact with a particular LDAP server. A SPI is a set of classes that implements various JNDI interfaces for a specific LDAP directory service. This flexibility provides a generic interface for Java applications to access LDAP directory services without considering provider specific details.

Next, we will illustrate, how a client application access a student entry in our LDAP DIT using the distinguished name.



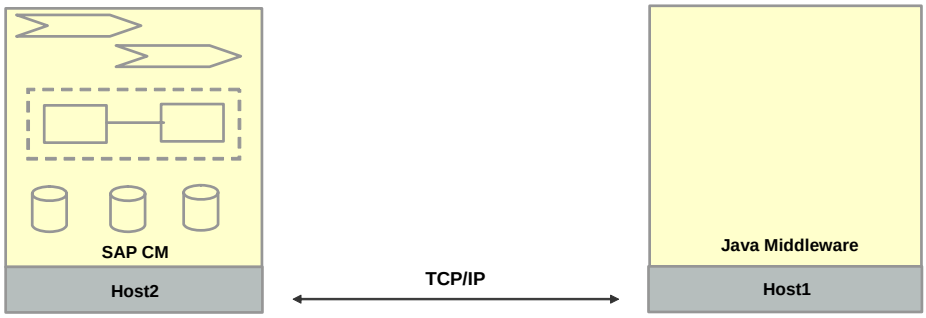
Information 31: Application Accessing Student Data with DN

Information 31 illustrates the accessing of student entries stored in the directory by a client application. The client application wants to get information of a student with the uid=123 from the directory. An interface in the application defines the methods, which access the directory. Through this interface the client calls a method requesting information of a specific student with its distinguished name uid=123, ou=Student, o=Phy, dc=uni-karlsruhe, dc=de. The interface in the client application calls generic methods in the LDAP API to retrieve the corresponding information. The API in turn calls a provider specific method in the SPI, then the SPI request the specific student entry using the same DN from the LDAP server. The LDAP server searches the DIT and return the information of the student with the DN. In turn the LDAP API returns this information of the student to the client application. In this way the client application can request and retrieve information of a specific student using a DN.

2.3 Interface to the Legacy System

In general there are different ways to import data into an SAP system. Here we want to explain the most common solution with our example. First we explain the different physical systems. As second we describe the data flow and show the architecture of the interfaces.

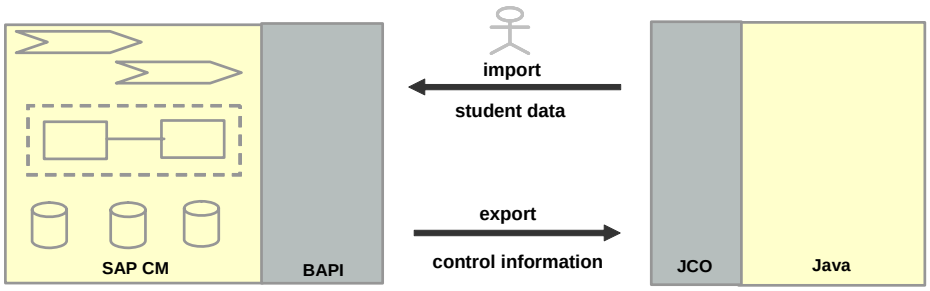
Location of the software



Information 32: Physical Separation of SAP and Middleware (Java/JCO)

The middleware uses Java with the SAP connector JCO. It is important to say, that the individual systems can be located on different computers. Merely an IP based internet connection must exist between the individual systems. TCP/IP is used as communication protocol. Java/JCO communicates via an own proprietary protocol with the BAPI. This communication is based on TCP/IP so that a simple internal connection can be used.

Data Flow between Middleware and SAP

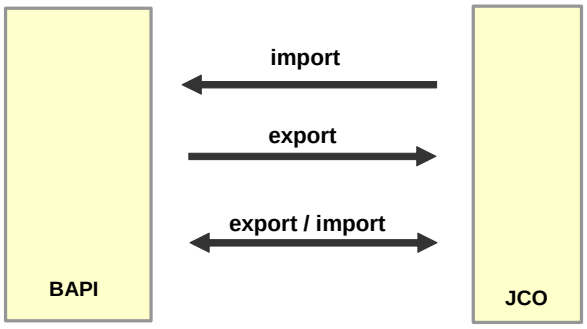


Information 33: Data flow between Java and SAP

In our scenario the student objects become imported from Java over the BAPI into the SAP system. At this new business objects are added when required into the SAP system. From the

perspective of the data flow, semantic information flows from the Middleware to the SAP system. At this it has to be importantly held tight that in the ideal case exclusively data sets are added in SAP. From SAP to Java only control information should flow, which gives the information, if the import process was successful or not. This ideal data flow isn't always realizable in the practice. E.g. it is possible, that an import without the student registration number takes place. Then the SAP system would set it automatically. In this case the question would be important, what kind of synchronization an import exactly represents. Because of the simplicity, we don't look at these difficulties.

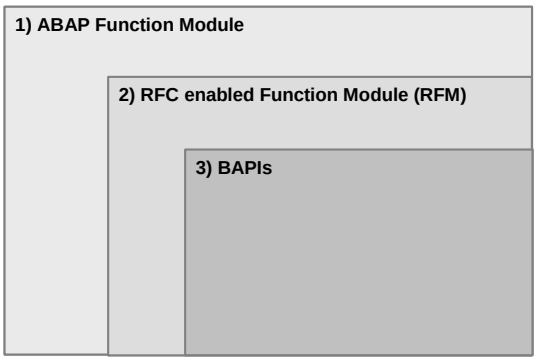
Interface between JCO and BAPI



Information 34: Technical Access Interface between JCO and BAPI

Between JCO and BAPI there are in principal two kinds of access methods. You can simply import or export data. As third opportunity you can also combine the import and export method. The transmitted data structure always depends on the interfaces the BAPI provides. In general the BAPI has a certain set of attributes. Each attribute can be a basic data type or it can be some kind of table, which includes a certain set of basic data types.

Architecture of BAPIs



Information 35: SAP Internal Structure of BAPIs, RFC and ABAP

Internally BAPIs consists of RFC enabled Function Modules and of ABAP Functions. The basic programming language is ABAP. The ABAP function modules, which can be used by different applications, are created as RFC enabled function modules. BAPIs are RFC enabled function modules, which can be accessed from an outstanding application in order to access the data of the SAP system. The BAPIs are a certain form of these RFC enabled function modules. Besides the detailed technical requirements, they have to satisfy further requirements regarding nomenclature and the documentation of their interfaces and their operation methods.

Syntactic Structure and Control Data of the Student Object

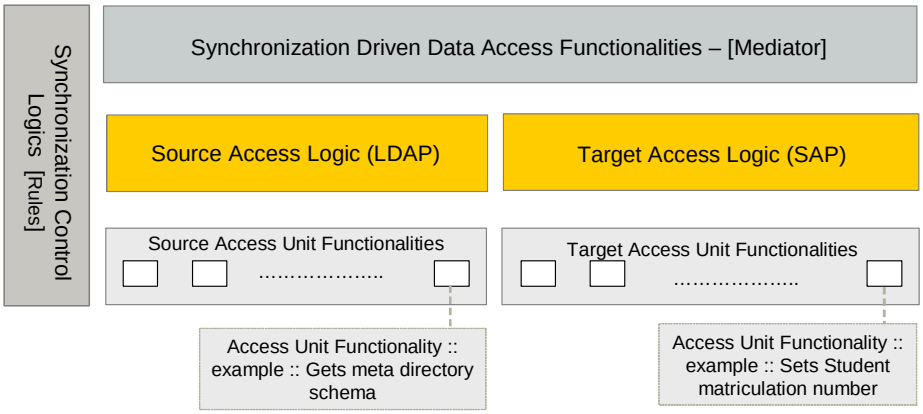
The syntactical structure describes the interface of the BAPI in greater detail. As mentioned before the BAPI BAPI_STUDENT_CREATEFORMDATA3 has a certain set of attributes, with different data types. Some of the data types are basic data types and some of them are kind of tables, which have basic data types integrated.

In addition to the already mentioned attributes, the BAPI itself, also have special attributes, which control the behavior of the BAPI. For example you can use it in order to start a test run, for testing the application. In the table below you see information about the control attributes.

Attribute	Description
STUDENTNUMBEREXTERN	Registration number of the student
Table	**Description**
TESTRUN	Changing on simulation mode at writing BAPIs
UPDATEACCOUNTDATA	Data item to the domain

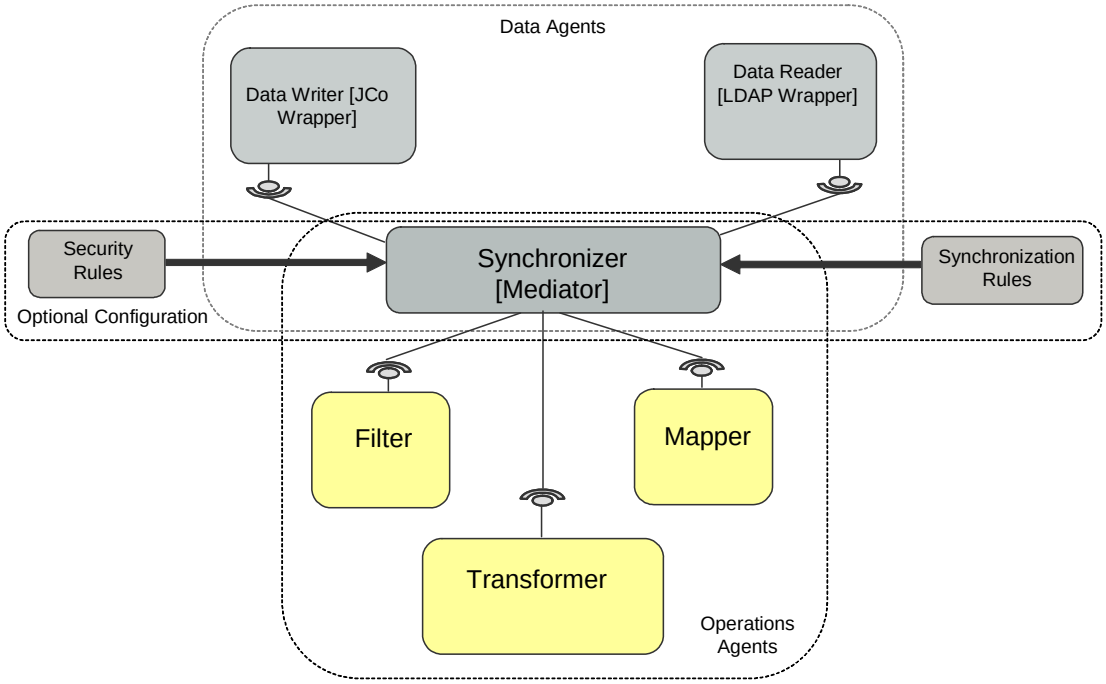
Through this additional control attributes, you can see, whether an import was successful or not. You can also use it in order to recognizing, which kind of error happened.

2.4 The Synchronization Application



Information 36: Design Phase – Synchronization: Data Access

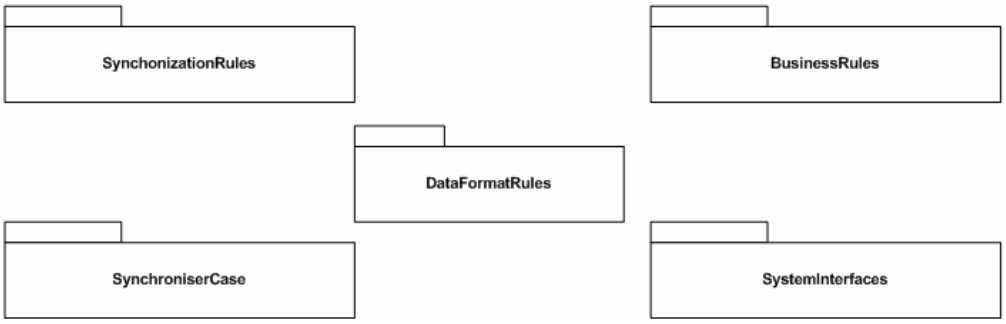
In
Information 36, we give another view of how data access is performed. The source Access Unit Functionalities build the set of base functionalities that can be combined together by respectively the Source and Target Access Logic layers to perform elaborated dialogue with the end-systems (meta-directory and the URP). One layer upper is the Mediator component that coordinates data access, offers an interface to the Core Synchronization Process, and provides necessary information to the Target Access Logic layers to perform their task (like access login and password or target organizational unit for example). These Three layers are themselves controlled through the transversal “Synchronization Control Logics” (“Rules”) layer, which can control any of the components on the three layers.



Information 37: Refined Synchronization Components Interfacing Model

Information 37 offers a refined view of how the already presented components collaborate together. For example the "Core Synchronization Process" components, the Transformer, the Mapper and the Filter offer an interface to be controlled by the mediator. The "Synchronization Rules" and "Security Rules" are components that permit configuration of the behaviour of the mediator. The "Data Reader" and the "Data Writer" components are called "Extraction Agents" and are exactly the "Target Access Logic" components that we've already seen, and which are specific to end-system collaborators.

In the following, we'll get to describe more practically our application architecture through some UML class diagrams. First of all the packages diagram is the following:



Information 38: Packages Diagram

Information 38 delivers more information about how the Core Synchronization components are splitted. We distinguish five elements.

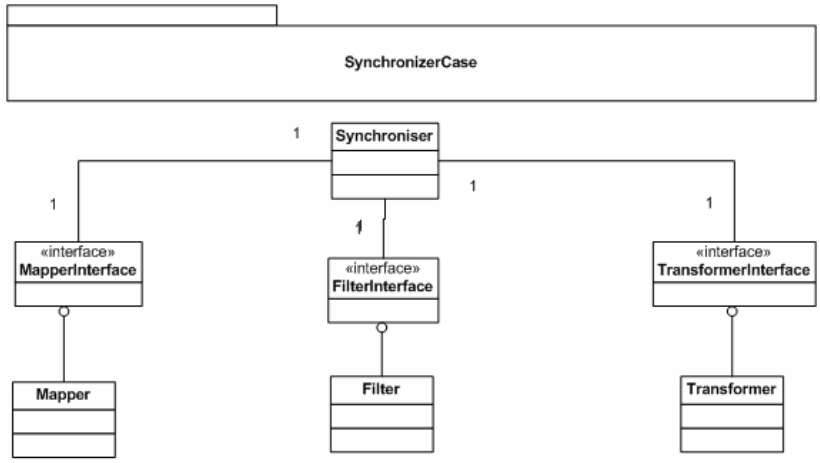
The "SynchronizationRules" package contains utilities that allow for delivery and setup information about synchronization procedures.
Example of a method: getMinAllowedRefreshTime(), delivers the minimum allowed lap time between two calls to a meta-directory synchronization. The use of these rules is compulsory.

The “DataFormatRules” package contains utilities that deliver information about desired data formats.
Example of a method: getStudentDestinationFieldList (), delivers the required fields to be accessible in the target repository data object.

The “SystemInterfaces” package contains utilities that allow for accessing the end-system collaborators.
Example of a method: storeResearcherNewInstitute(), stores a new attachment institute for a researcher.

The “SynchroniserCase” package contains utilities that perform the Core Synchronisation Process operations.
Example of a method: filterStudentInformation(), filters irrelevant information from the student data object.

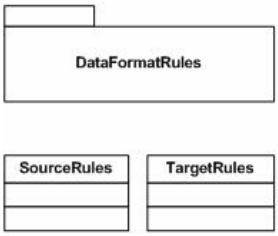
The “BusinessRules” package contains utilities that allow for managing how the Core Synchronisation Process components operate. This package has been viewed as a further development step and is unlikely to be developed as part of the initial increments.



Information 39: Synchronizer Classes

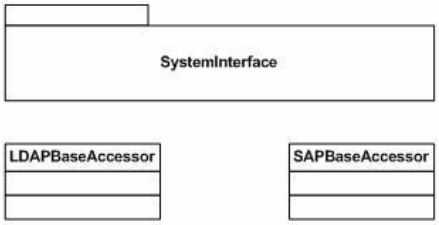
In Information 39: Synchronizer Classes are presented the SynchronizerCase classes. There are three interfaces that correspond to the three base synchronization tools:

- The Mapper :takes in charge operations that seek to retain just data that can be stored in the Student BAPI
- The Filter :contains operations whose goal is to process the needed operations on data in order to make it exploitable from the Student BAPI
- The Transformer: allows for complex transformation on data structures, for example when a complex structure is needed to be constructed, and/or some of its fields must be compute or elaborated through specific mechanisms, in order to be accepted by the end-system, in our case the tudent BAPI (one can imagine a non-trivial BAPI, that needs a table to be constructed, some fields of which have to be computed and can not be deduced through simple associations, between the values in the LDAP server and the required values in the SAP BAPI).
- The Synchronizer contains the logic that makes the right flow of calls to the mechanisms offered by the three base classes, seeking to make the storage of the synchronized data a success.



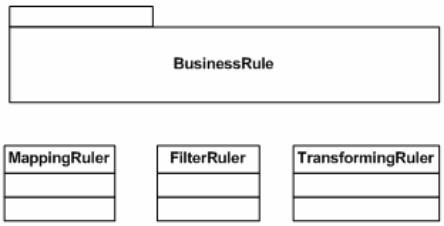
Information 40: Data Format Rules

The Information 40: Data Format Rules shows the components of the DataFormatRules package, the SourceRules class and the TargetRules class. Both classes can be consulted to deliver information about the allowed or required operations before any call to the SynchronizerCase Package classes.



Information 41: System Interfaces

The SystemInterface package contains the needed classes for accessing the data resources (LDAP and BAPI).



Information 42: Business Rules

The BusinessRules Package contains respectively the mapping rules, the filtering rules and the transforming rules. These are optional tools that can be used to easen the programming of the synchronization process, in the way they can deliver mapping, filtering or transformation logics,automatically, and thus we wouldn't need to include this part of the logics in the implementation code of the SynchronizerCase package classes. They are a sort of mapping, filtering or transformation templates. An example to this would be that the MappingRuler class could include a method that returns a list of student data elements that can be extracted from the LDAP server, and that can be directly mapped to required values in the SAP student BAPI, also indicating which are the target values to be matched with which source values.

2.5 Exercises

(1) Explain how a java application communicate with a LDAP directory service

(2) How will data from the source and in the target repositories be accessed ?

(3) What utility classes are used and managed by the data synchronizer ?

(4) How is the location of Java/JCO, Middleware and SAP R/3 organized in a typical synchronization scenario.

(5) Why do we also have to export data from the SAP system in the context of our synchronization process ?

(6) What kind of data must be exported ? Why ?

Information 43: Exercises

(1) Information 30: LDAP Communication with a Java Application

(2) Information 36: Design Phase – Synchronization: Data Access

(3) Information 37: Refined Synchronization Components Interfacing Model

(4) Information 32: Physical Separation of SAP and Middleware (Java/JCO)

(5) Information 33: Data flow between Java and SAP

(6) Information 33: Data flow between Java and SAP

3 IMPLEMENTATION, DEPLOYMENT AND USAGE

3.1 Implementation

3.1.1 LDAP Side

JNDI vs. JLDAP

In this paragraph we will try to show why we have chosen which API in order to access the LDAP server. Since we decided to implement the synchronization application in JAVA, we had the choice between two APIs, JLDAP by NOVELL and JNDI by SUN. What exactly are those two tools and how do they work? Moreover, why did we choose JNDI rather than the JLDAP? These are the questions we'll be trying to answer in this paragraph. Lightweight Directory Services (LDAP) has already been introduced in this case study.

JLDAP

First we'll get to know a little bit more about JLDAP, the Java development classes for LDAP, from NOVELL. Since NOVELL is a major contributor to the OpenLDAP project, the JLDAP foundation classes are indicated on the official website of the OpenLDAP project. LDAP Classes for Java enable to write applications that access, manage, and update information stored in NOVELL eDirectory or other LDAP-aware directories, such as NDS, Microsoft's Active Directory or OpenLDAP. JLDAP is designed to provide powerful, yet simple, access to LDAP directory services. This API defines both asynchronous and synchronous interfaces to LDAP to suit a wide variety of applications.

Brief Introduction

Operations are provided to authenticate, search for and retrieve information, modify information, and add and delete entries from the tree. An LDAP server may return referrals if it cannot completely service a request (for example if the request specifies a directory base outside of the tree managed by the server). JLDAP offers the programmer three options: the programmer can catch these referrals as exceptions and explicitly issue new requests to the referred-to servers, the programmer can provide an object to establish a new connection to a referred-to server, or the programmer can let the library automatically follow the referrals. In the latter case, the programmer may also provide a re-authentication object, allowing automatic referrals to proceed with appropriate credentials. If no such object is provided, referrals are followed with anonymous credentials, and the protocol level of the original connection is used. Before the client encodes and sends a string value to a server, the string values are converted from the Java 16-bit Unicode format to UTF-8 format, which is the standard string encoding for LDAPv3 servers. The integrity of double-byte and other non-ASCII character sets is fully preserved.

Overview of JLDAP

The central LDAP class is LDAPConnection. It provides methods to establish an authenticated or anonymous connection to an LDAP server, as well as methods to search for, modify, compare, and delete entries in the directory.

The LDAPConnection class also provides fields for storing settings that are specific to the LDAP session (such as limits on the number of results returned or timeout limits). An LDAPConnection object can be cloned, allowing objects to share a single network connection but use different settings (using LDAPConstraints or LDAPSearchConstraints).

A synchronous search conducted by an LDAPConnection object returns results in an LDAPSearchResults object, which can be enumerated to access the entries found. Each entry (represented by an LDAPEntry object) provides access to the attributes (represented by LDAPAttribute objects) returned for that entry. Each attribute can produce the values found as byte arrays or as Strings.

Overview of LDAP API Use

- An application generally uses the LDAP API in four steps.
- Construct an LDAPConnection. Initialize an LDAP session with a directory server. The LDAPConnection.connect() call establishes a handle to the session, allowing multiple sessions to be open at once, on different instances of LDAPConnection.
- Authenticate to the LDAP server with LDAPConnection.bind().
- Perform some LDAP operations and obtain some results. The synchronous version of LDAPConnection.search() returns an LDAPSearchResults which can be enumerated to access all entries found. The asynchronous version of LDAPConnection.search() returns an LDAPSearchListener, which is used to read the results of the search. LDAPConnection.read() returns a single entry.
- Close the connection. The LDAPConnection.disconnect() call closes the connection.

Asynchronous & Synchronous Operations with JLDAP

There are both synchronous and asynchronous versions of the LDAP protocol operations in this API.

- Synchronous methods do not return until the operation has completed.
- Asynchronous methods take a listener parameter (either LDAPResponseListener or LDAPSearchListener) and return a listener object which is used to enumerate the responses from the server. A loop is typically used to read from the listener object, which blocks until there is a response available, until the operation has completed.
- An LDAPResponseListener may be shared between operations, for multiplexing the results. In this case, the object returned on one operation is passed in to one or more other operations, rather than passing in null.

Notice about LDAP referrals: An LDAP referral contains a list of one or more URLs. To process an LDAP referral, the service provider uses the information in these URLs to create connections to the LDAP servers to which they refer. Multiple LDAP or LDAPS URLs in a single referral are treated as alternatives, each followed until one succeeds. The complete URL (including any query components) is used.
You set up referrals by creating *referral* entries in the directory that contain the `"REF"` attribute. This attribute contains one or more referral URLs (usually LDAP or LDAPS URLs)

Directory Services Markup Language – DSML

Directory Services Markup Language, or DSML, an OASIS specification, enables developers to express LDAP functions and retrieve data in XML. DSML version 2 (DSMLv2) represents LDAP directory operations and their results by XML request/response operations. Common DSML operations include searching for specific directory objects and returning selected attribute values. This is very interesting when developing using XML and SOAP, DSML makes it very easy to integrate with the directory using the tools you know. Thus LDAP services can be integrated within an SOA (Service Oriented Architecture. It is supported by the

JNDI

JNDI stands for Java Naming and Directory Interface. It is a product developed by SUN. Many aspects distinguish JNDI from other LDAP programming tools under JAVA. We'll first have a

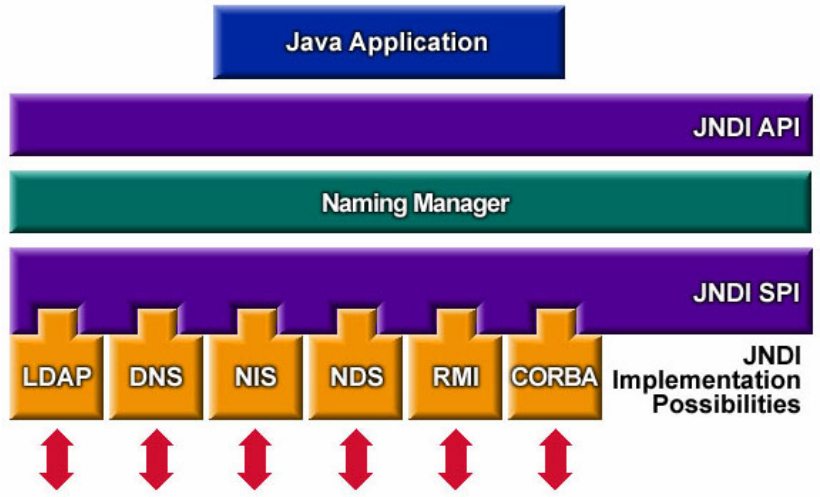
look at how it is designed and its properties, and then try to justify the choice made to use JNDI rather than other tools.

Brief Introduction

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) that provides naming and directory functionality to applications written using the Java programming language. It is defined to be independent of any specific directory service implementation. Thus a variety of directories - new, emerging, and already deployed - can be accessed in a common way. Thus it is very different in its conception than the JLDAP. JNDI provides a way of connecting to various kinds of services that are indexed in a registry. JNDI is modeled on LDAP (Lightweight Directory Access Protocol) and usually associated with LDAP. But importantly, JNDI also forms the backbone of other important APIs, including RMI, EJB, JMS, and CORBA.

Overview of JNDI

JNDI allows applications to access various naming and directory services via a common interface. Like JDBC (Java Database Connectivity), JNDI is not a service, but a set of interfaces. It allows applications to access many different directory service providers using a standardized API. Just as with JDBC, the JDK contains the JNDI interfaces but does not include a JNDI service provider although Sun Microsystems provides adapters for connecting to existing directory service providers, such as LDAP. However, one can use one of several free or open source JNDI providers in the J2SE (Java 2 Platform, Standard Edition) applications.



Information 44: JNDI Architecture

In the Information 44: JNDI Architecture, it is shown that the JNDI architecture consists of an API and a service provider interface (SPI). Java applications use the JNDI API to access a variety of naming and directory services. The SPI (Service Providers Interface) enables a variety of naming and directory services to be plugged in transparently. Thereby it allows Java applications using the JNDI API to access their services. It is therefore very adaptable and the implementation possibilities make it very flexible and to be integrated in heterogeneous contexts where many directory and naming services co-exist.

JNDI in J2EE and J2SE

The Sun documentation for JNDI indicates the following:
"The JNDI is included in the Java 2 SDK, v1.3 and later releases. It is also available as a Java Standard Extension for use with the JDK 1.1 and the Java 2 SDK, v1.2. It extends the v1.1 and v1.2 platforms to provide naming and directory functionality.
To use the JNDI, you must have the JNDI classes and one or more service providers. The Java 2 SDK, v1.3 includes three service providers for the following naming/directory services:
Lightweight Directory Access Protocol (LDAP)
Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service
Java Remote Method Invocation (RMI) Registry"
[S-JNDI]

(1) javax.naming

(2) javax.naming.directory

(3) javax.naming.event

(4) javax.naming.ldap

(5) javax.naming.spi

Information 45: Packages of JNDI

The JNDI is divided into five packages:

- javax.naming: package contains classes and interfaces for accessing naming services.

- javax.naming.directory: The javax.naming.directory package extends the javax.naming package to provide functionality for accessing directory services in addition to naming services. This package allows applications to retrieve attributes associated with objects stored in the directory and to search for objects using specified attributes.

- javax.naming.event: As the naming/directory service plays an increasingly important role in the computing environment, the need to provide administration and monitoring tools to help manage changes in the service also increases. For such tools and other applications, the traditional request/response style of interaction needs to be augmented with an asynchronous notification model that allows applications to register interest in changes in the service. The javax.naming.event package contains classes and interfaces for supporting event notification.

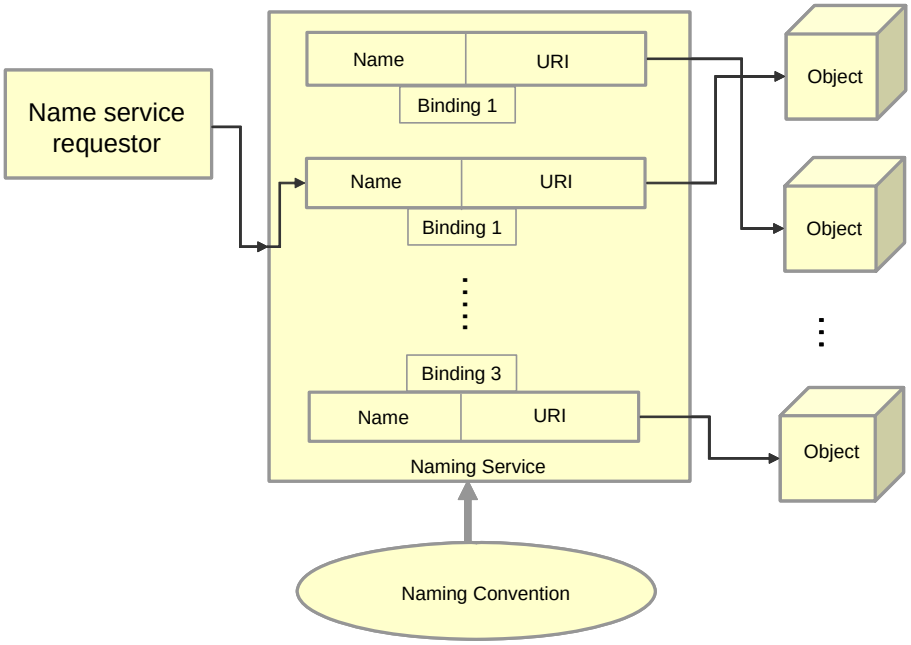
- javax.naming.ldap: This package contains classes and interfaces for using features that are specific to the LDAPv3 that are not already covered by the more generic javax.naming.directory package. In fact, most JNDI applications that use the LDAP will find the javax.naming.directory package sufficient and will not need to use the javax.naming.ldap package at all. This package is primarily for those applications that need to use "extended" operations, controls, or unsolicited notifications.

- javax.naming.spi: This package provides the means by which developers of different naming/directory service providers can develop and hook up their implementations so that the corresponding services are accessible from applications that use the JNDI.

Service Providers and Naming Services

Notice about service providers: JNDI can be implemented in almost any hierarchical naming structure. An implementation of JNDI for a particular environment is called a Service Provider. And to use JNDI in a particular environment you need the Service Provider for that environment.

Notice about naming services: we just talked about naming services, so let's just shortly tell what it is and what the use of it is. Finding resources is of particular importance in large-scale enterprise environments, where the applications you build may depend on services provided by applications written by other groups in other departments. A well-designed naming infrastructure makes such projects possible -- and the lack of one makes them impossible. In fact, many business-process reengineering efforts begin with the design and implementation of a robust, enterprise-wide naming and directory infrastructure. A naming service maintains a set of bindings.



Information 46: Naming Service Example

JNDI was not designed to replace existing technology; instead, it provides a common interface to existing naming services (JNDI presents an abstraction over all these different services), it is since very different from JLDAP. Let's begin by taking a look at some of these services. Some types of Naming services other than LDAP are:

- COS (Common Object Services) Naming: The naming service for CORBA applications; allows applications to store and access references to CORBA objects.
- DNS (Domain Name System): The Internet's naming service; maps people-friendly names (such as www.etcee.com) into computer-friendly IP (Internet Protocol) addresses in dotted-quad notation (207.69.175.36). Interestingly, DNS is a *distributed* naming service, meaning that the service and its underlying database is spread across many hosts on the Internet.
- NIS (Network Information System) and NIS+: Network naming services developed by Sun Microsystems. Both allow users to access files and applications on any host with a single ID and password.

Directory Services Markup Language - DSML

JNDI allows as well to work with DSML. The official JNDI documentation specifies that JNDI includes a service provider for DSML. The JNDI/DSML v2.0 service provider enable access to DSML v2.0 services that are implemented as a Web service over SOAP/HTTP 1.1 or as a static document (DSML file).

Security on the LDAP

To access the LDAP service, the LDAP client first must authenticate itself to the service. The LDAP client must tell the LDAP server who is going to be accessing the data so that the server can decide what the client is allowed to see and do. If the client authenticates successfully to the LDAP server, then when the server subsequently receives a request from the client, it will check whether the client is allowed to perform the request. This process is called access control.
The LDAP standard has proposed ways in which LDAP clients can authenticate to LDAP servers (RFC 2251 and RFC 2829).

- The LDAP standard proposes several authentication possibilities:
- Anonymous authentication – is a non-authenticated access to the server, lets the LDAP client only read public data.
- Root DN Authentication – is the privileged user. He has modifying access rights on all data.
- Unencrypted password – The password is transmitted unencrypted on the network. Is ideally used on a sub-network protected by a firewall and where data is not sensitive.
- Password + SSL or TLS – session between a server and a client, the password is encrypted.
- Certificates + SSL – Simple Authentication and Security Layer (SASL) – enables to call for more elaborated key-based authentication mechanisms (OTP, Kerberos...)

Another security aspect of the LDAP service is the way in which requests and responses are exchanged between the client and the server. Many LDAP servers support the use of secure channels to communicate with clients, for example to send and receive attributes that contain secrets, such as passwords and keys. With LDAP servers SSL is often used for this purpose.

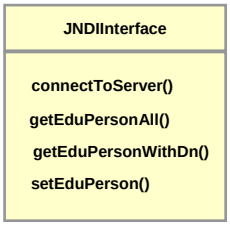
Notice about client requirements: The LDAP service provider uses the Java Secure Socket Extension (JSSE) software for its SSL support. The JSSE is available as part of the Java 2 SDK, v1.4.

Conclusion

As we have seen, there are pretty much a reasonable amount of differences between the JLDAP and the JNDI. We have tried to put in perspective the stakes of using the JNDI and have identified some properties that make the balance lean to the JNDI side. Of course this choice has to be put in the perspective of a complete, flexible and configurable implementation of the synchronisation application. Which as we'll see is not the case in this implementation report, since we have decided to program a demonstration application that shows how a single dump of an LDAP server student information content, could be properly extracted and migrated to the SAP system.

LDAP Wrapper Functionalities

The LDAP Wrapper provides the functionality to any client application, in our system the Synchronizer application, to access data stored in the LDAP Server without considering any implementation detail. It encapsulates physical aspects from the logical and business aspects and provides a consistent interface to the Synchronizer application. Therefore, the Synchronizer can easily access student data stored in the LDAP Server by using the functionalities provided by the LDAP Wrapper. Now we will briefly introduce the functionalities of the LDAP Wrapper.



(1) connectToServer(): Establish a connection with the LDAP Server

(2) getEduPersonAll(): Retrieves all the objects stored in the directory

(3) getEduPersonWithDn(): Retrieves a specific object using it's DN

(4) setEduPerson(): Specifies which type of objects should be retrieved

Information 47: JNDI Wrapper

The LDAP Wrapper or the interface to the LDAP Server provides four functionalities. First it allows the Synchronizer application to establish a connection with the LDAP server. It gives the application the possibility to define parameters at run time, so that it can define the server it would like to establish a connection with as well as its root DN. It also provides the functionality to retrieve all the objects of a particular type, for example students, in the directory. This is one of the main functions in our system, since we have to synchronize student data stored in an LDAP directory and the SAP System. After establishing the connection with the server the Synchronizer application can use this method to retrieve all the student objects in the directory. Additionally we provide two more methods in our LDAP Wrapper, one of them provides the functionality for any application to access a specific object in the directory using its DN. For example using the DN, "UserId=123123123,ou=Student,o=Informatik,dc=uni-karlsruhe,dc=de", the application can retrieve a specific student object in the directory with the specific DN. As we have already mentioned in earlier chapters, our directory does not have only student information, rather it has information regarding all the people involved with the university. Therefore, we must provide the client application the possibility to define at run-time which type of objects (students or staff) it would like to retrieve or manipulate. The last method in the LDAP Wrapper provides this functionality, so that an application can define at run-time, which objects in the directory it would like to deal with.

The LDAP Wrapper functionalities, establishing a connection with the LDAP Server and accessing objects in the directory are realized using JNDI and in the next section we will present a brief overview of the implementation of the LDAP Wrapper using JNDI.

Implementation of LDAP Wrapper using JNDI

In this section, we will briefly describe the implementation of the LDAP Wrapper we have introduced in the above section. The implementation of the LDAP Wrapper is mainly realized using JNDI and here we will present some of the core functionalities of JNDI we have employed in order to access the student information in the directory.

Establishing a Connection with the LDAP Server

There are several ways in which a connection is created. The most common way is from the creation of an initial context. In JNDI all naming operations are relative to a context. The initial context implements the Context interface in JNDI (This interface represents a naming context, which consists of a set of name-to-object bindings. It contains methods for examining and updating these bindings) and provides the starting point for resolution of names. When the initial context is constructed, its environment is initialized with properties defined in the environment parameter passed to the constructor. When you create an initial context by using

the LDAP service provider, a connection is set up immediately with the target LDAP server named in the environment properties.

```
public void connectToServer()
{
    String ldapServerName = “localhost“;
    String rootdn       = “cn=Manager,dc=uni-karlsruhe,dc=de“;
    String rootpass      = “*******“;
    String rootdn       = “dc=uni-karlsruhe,dc=de“;

    Properties env = new Properties();

    env.put( Context.INITIAL_CONTEXT_FACTORY,"com.sun.jndi.ldap.LdapCtxFactory" );
    env.put( Context.PROVIDER_URL, "ldap://" + ldapServerName + "/" + rootContext );
    env.put( Context.SECURITY_PRINCIPAL, rootdn );
    env.put( Context.SECURITY_CREDENTIALS, rootpass );

    try {
        dircontext = new InitialDirContext(env);
    } catch ( NameAlreadyBoundException nabe ) {
    System.err.println( "value has already been bound!" );
    }
}
```

LDAP Server Parameters

Create Environment

Initilize Environment

Obtain initial context using the environment/connect to server

Information 48: Connecting to LDAP Server

Our LDAP Wrapper establish a connection with the server by creating an InitialDirContext and InitialDirContext is an initial context which implements the Context interface and additionally it also implements the DirContext (The directory service interface, containing methods for examining and updating attributes associated with objects, and for searching the directory). First it initializes an environment with its environment properties. In our case we initialize the environment with four parameters that are defined in the Context interface. The first parameter INITIAL_CONTEXT_FACTORY holds the name of the environment property for specifying the initial context factory to use. The value of the property should be the fully qualified class name of the factory class that will create an initial context. Since we access an LDAP directory service it is "com.sun.jndi.ldap.LdapCtxFactory". The second parameter PROVIDER_URL holds the name of the environment property for specifying configuration information for the service provider to use. The value of the property should contain a URL string (e.g. "ldap://localhost:389"). The third parameter SECURITY_PRINCIPAL holds the name of the environment property for specifying the identity of the principal for authenticating the caller to the service, in our system it is the root DN ("cn=Manager,dc=uni-karlsruhe,dc=de") and the last parameter SECURITY_CREDENTIALS holds the name of the environment property for specifying the credentials of the principal for authenticating the caller to the service. The value of the property depends on the authentication scheme. For example, it could be a hashed password, clear-text password, key, certificate, and so on.

After the environment is initialized with its properties, it is passed to the constructor of the InitialDirContext, where we obtain an initial context which allows us to perform naming operations and automatically set up a connection with the target LDAP server.

Since we have described how we obtain an initial context to perform naming operations, now we will introduce how we access information in the directory using the initial context.

Searching the Directory

One of the most useful features that a directory service offers is its search service. You can compose a query consisting of attributes of entries that you are seeking and submit that query to the directory. The directory then returns a list of entries that satisfy the query. In our system, we also need to search the directory, since we are synchronizing student data between the directory and another system, often we need the query “Retrieve all student objects in the directory”. These student objects may be placed under different sub-trees, for example students in different faculties. Therefore, the search should give us all the student objects without considering where they reside in the directory. Now we will briefly explain how we realize this directory search using JNDI.

The DirContext interface in JNDI provides several methods for searching the directory, with progressive degrees of complexity and power. Since the initial context (InitialDirContext) we obtain, when we connect to the server implements the DirContext interface, we can use the initial context to perform the required search. Although it provides several search methods, in our discussion we will only introduce the search method we have employed to search all the student objects.

```
SearchControls ctls = new SearchControls();

ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);

String filter = "(deEduPersonMatrNum=*)";

try
    {
     NamingEnumeration answer = dircontext.search("", filter, ctls);

     }catch(javax.naming.NamingException ne)
     {
        System.out.println(ne.toString());
     }
```

Initialize search controls

Set search controls

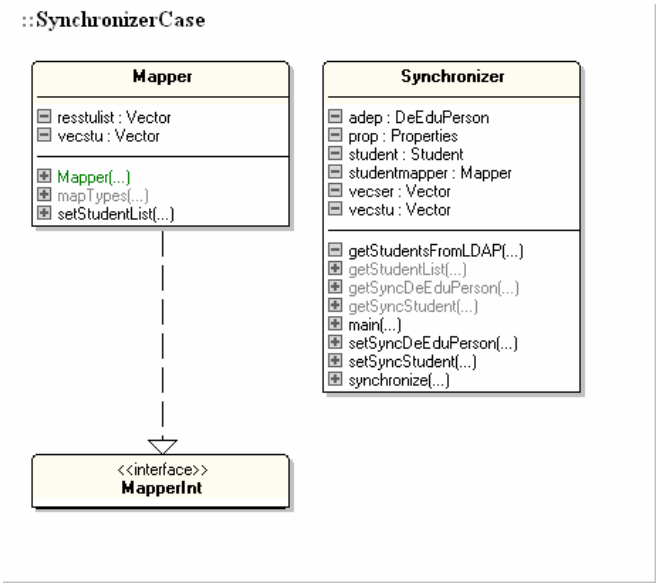
Set search filter

Perform Search

Information 49: Searching the Directory

The search method we have employed from the DirContext API requires three parameters. The first parameter defines the name of the target context in which to perform the search. Here we have defined it as “”, i.e. the search starts from the root of the initial context (“dc=uni-karlsruhe,dc=de”) we have specified. For example, if we set this parameter as “o=Informatik”, then the search starts from “o=Informatik”in the directory tree. The second parameter is in the form of a search filter. A search filter is a search query expressed in the form of a logical expression. Since we are searching only student entries in the directory, our search filter "(deEduPersonMatrNum=*)" specifies that the qualifying entries must have an attribute deEduPersonMatrNum. The third parameter is of the type SearchControl, which controls search aspects such as, attributes to return, scope of search and maximum number of entries to return. We have used this parameter to define the scope of the search, the constant SUBTREE_SCOPE specify that the search be performed in the entire sub tree. The search results we get are collected in a NamingEnumeration for further processing.

3.1.2 Synchronizer Side

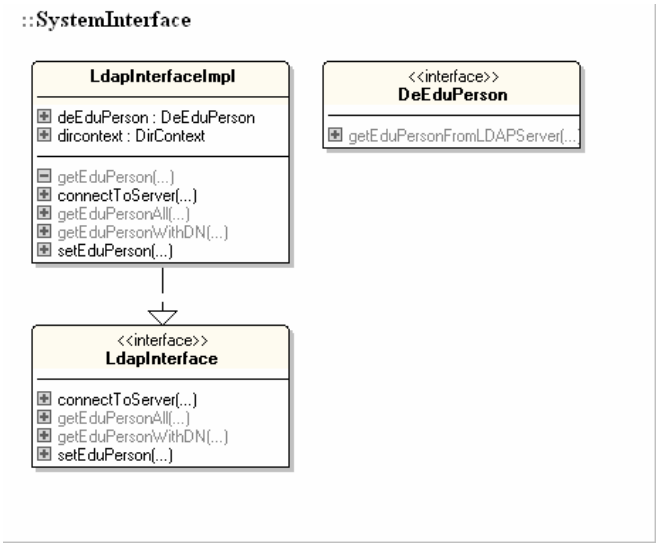


Information 50: The Synchronizer Class Implementation

The implemented synchronization application has been made by programming the "Synchronizer.java" class. This class executes the fundamental following operations:

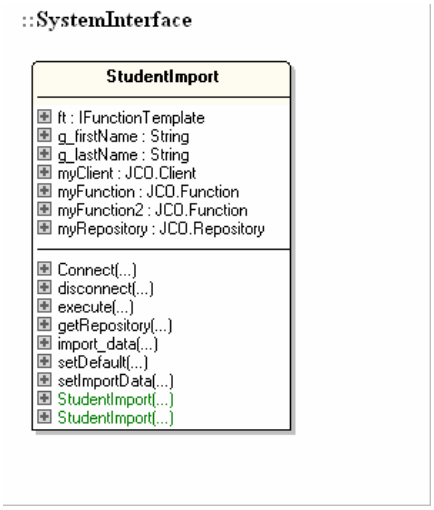
- Instanciates a new ldap wrapper
- Instanciates a new student LDAP data mapping object ("class Student.java"), which is our student data persistence class
- Connects to the LDAP server
- Calls the extraction of all students data
- Instanciates a new mapper
- Operates mapping operations on the extracted data
- Instanciates a new SAP wrapper
- Iterates through the extracted mapped objects and calls the SAP import wrapper to store each one of these mapping objects

Here are UML Class Diagrams for the LDAP and the SAP wrapper classes, for a better understanding of the provided functionalities:



Information 51: The LDAP Export Wrapper Class Implementation

Information 51: The LDAP Export Wrapper Class Implementation is a class diagram representing the LdapInterfaceImpl class.

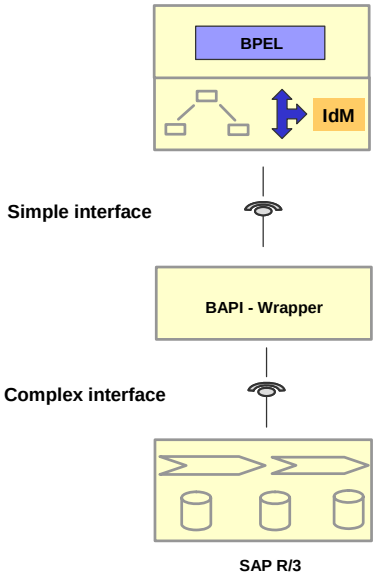


Information 52: The SAP Import Wrapper Class Implementation

Information 52: The SAP Import Wrapper Class Implementation is a class diagram of the StudentImport class, which is responsible for the storage of data in the SAP student BAPI

3.1.3 SAP Side

Reasons for Using a Java/JCo Wrapper in General



Information 53: Advantages of a Java/JCO – Wrapper

Why is it meaningful to integrate a Wrapper Interface in order to write data into a SAP R/3 legacy system. Or what problems can one have, if he does not use such a Wrapper interface. As you can think, there are many reasons do implement the architecture this way.

Specific Access Rights have to be taking into Account

In order to write data into an existing legacy system, the necessary access rights have to be granted to the middleware application. Because of this there are several configuration files, which have to be stored inside the wrapper. This can possibly be done through a configuration file, which can be read from the Wrapper class. There is a lot of data needed for the configuration. For example the Wrapper needs the information, where the BAPI - Server is located. And the login data is very important. There may also be different configuration files between different legacy systems or different business scenarios. All this different kind of configuration mechanism can be implemented inside the Wrapper. Because of this the interface to the legacy system is more or less the same for the application developer, so that he can concentrate on the core development.

Transaction Behavior

The interfaces to a legacy system can be seen as database similar. Because of this some of them also have methods, which are very similar to the transaction model of relational databases. After writing data into a BAPI, this has to be committed, and if an error happened or if some of the data have unexpected values, it has to be rolled back. Inside the application this complexity of access methods are disturbing and increases the effort of the core development very much.

Complexity of Interfaces

The interfaces are sometimes very complex. And it can be a very time consuming task to realize this task. There are several special boundary conditions, which have to be taken into account in order to offer a read and write interface. For every legacy system the developers need special knowledge. There are several people, who have certain knowledge of the different legacy system. With a wrapper module, one can define the interfaces for one legacy system.

Afterwards a different developer with different knowledge can do this for another legacy system.

Abstraction of Legacy System specific Problems

Through the Wrapper, the access of the legacy system becomes much easier. Some of the legacy system specific problems does not touch the application logic in principal. All of the specific problems of one legacy system are hided between the interfaces of the Wrapper. For example the data, which have to be imported or exported, belongs to different BAPIs. Although this data only represents one business object inside the application logic. Or the application has to be notified in real-time, if some of the data flags inside the legacy systems changes.

Abstraction of Synchronization Logic

If data in a user scenario is stored redundantly, the synchronization process needs to know what flags and attributes have changed, since the last synchronization. This kind of status monitoring and saving what has changed, should also be done inside the Wrapper. For example, the application programmer only writes or updates data of an existing business object in SAP. The Wrapper then checks, what has to be updated and which attributes have to be left with the same value.

Exchange of the Legacy System

The Wrapper can be exchanged with another Wrapper. This offers a very good flexibility, if someone wants to use the same application with a different legacy system. One can simply exchange the Wrapper with another Wrapper, who has interfaces to this legacy system. In this case the new Wrapper has to transform the relevant data in a way, that it is compatible to the attribute types of the new legacy system. The transformation from the attribute types of the old legacy system to the attribute types of the new legacy system has to be done inside the new Wrapper. Except that nothing else has to be changed inside the application logic.

Wrapper Framework for fast Prototyping

After getting some knowledge of the architecture of the legacy system interface an the way how to access them from the middleware, one has a good starting point for developing a general framework to access other kind of legacy system interfaces. With a certain type of legacy system Wrapper Framework, one can very fast develop a prototype for software solutions, which access the legacy system.

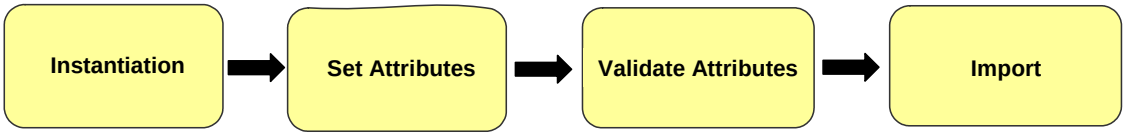
Better Representation of the needed Business Objects

The Wrapper can represent the specific business objects in a better way. For example each Wrapper imports a business object, which is needed in the application logic. It doesn't matter how the data is imported into the legacy system. From the point of an application developer, there is only the object he wants to integrate. This improves the modularity and the object oriented design of the application.

Maintenance

The object oriented design, the attributes of the used business objects and the hided interfaces to the legacy system decreases the complexity of the software and makes it easier to understand. So it is easier for an unknown developer to understand and adapt the behavior of the system. Because of this maintenance become much easier and faster. The effort for adapting the software to new versions of the legacy system decreases and the maintenance cost is much lower than before.

Process Sequence before the Import



Information 54: Process Sequences at the Import

Before the real import the application developer has to set some special attributes in order to define the relevant information for the import. In order to do this, there has to be a sequence of tasks, which have to be done in order to realize this.

First the Wrapper class has to be instantiated by the core java application. This instantiation should be a representation of the business object that has to be imported. This can be for example, the equivalent object to the SAP BAPI. It possibly has the same attributes and it should have a method, which offers the opportunity to set a minimum set of attributes, which are needed for the import process.

If the type of values, which have to be set, are very complex, some kind of validation method can be very helpful in order to avoid a wrong import. During the import into the legacy system, there also must be a validation. So one can first drive a test run in order to see whether the import data is syntactically correct or not. This concept has the advantage, that the validation itself doesn't have to be implemented.

At the end, the import itself has to be done and the return values of it has to be return back to core java application in order to give back the control information whether the import was successful or not.

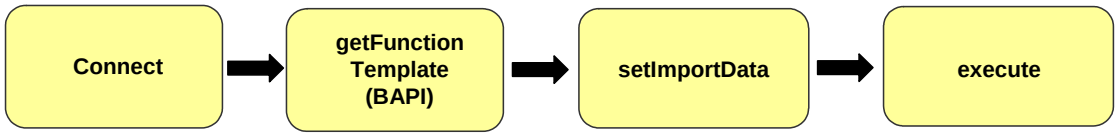
Used BAPI's

There are at least 2 BAPI's, which have to be used in the context of our import process. First of all we have to start the BAPI, BAPI_STUDENT_CREATEFROMDATA3 in order to import the student. After a successful import of this student information, we have to commit this student data by the BAPI BAPI_TRANSACTION_COMMIT. This commit BAPI is BAPI, which have to be used for a lot of other BAPI's. It can be seen as a similar function to the SQL-statement commit. If one needs to set additional information through another BAPI, which belongs to the newly created student, one can first create this student, set the additional data and afterwards finish it through a commit. In some cases, this can be very important. For example if you have to guarantee that creating a student and setting additional data has to be an atomic operation.

ABAP Implementation

The two BAPI's BAPI_STUDENT_CREATEFROMDATA3 and BAPI_TRANSACTION_COMMIT has to be called sequentially inside a logical function module. If you call two different BAPIs remotely over the JAVA/JCO interface, you don't have a logical function module. If you do so, then the creation of the students does not have a corresponding commit. That means that the students won't be created inside the SAP R/3 system. In order to realize this, we have to develop a remote possible function module, which calls both functions. We named this BAPI Z_CM_STUDENT_PROVISIONING. Inside this function we first call BAPI_STUDENT_CREATEFROMDATA3 and then BAPI_TRANSACTION_COMMIT. The relevant source code of the Z_CM_STUDENT_PROVISIONING implementation can be found in the Appendix.

JCO Implementation



Information 55: JCO Implementation, Access to SAP R/3

The JCO – implementation first connects to the BAPI server. After the connection the Java/JCO middleware has to log in to SAP R/3. Afterwards one have to get the Function Template (The used BAPI). After receiving the function template, the needed attribute have to be set. After setting the needed attribute, the middleware has to start the execution has to take place.

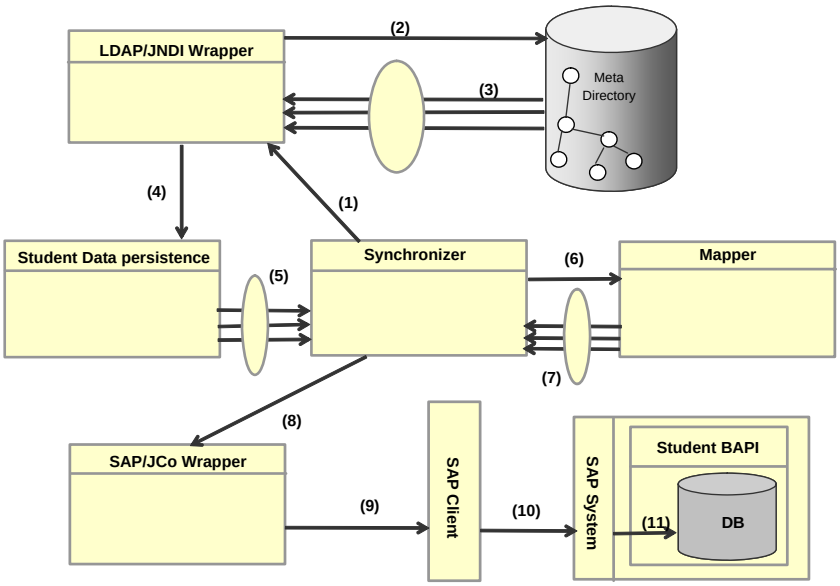
3.2 Increment Definition

In this paragraph we will present the chosen set of designed functionalities that we have decided to implement, in order to obtain a first version oft the application, this is what will be called, the first increment.

What is critical in the application is the way it interacts with the two end systems, namely the LDAP server and the SAP system. This is why our first concern was to obtain a first functional version of the LDAP (JNDI) and SAP (JCo) wrappers. We wanted thus to perform a minimal task and to be able to offer a demonstration of a subset of the initially designed functionalities.

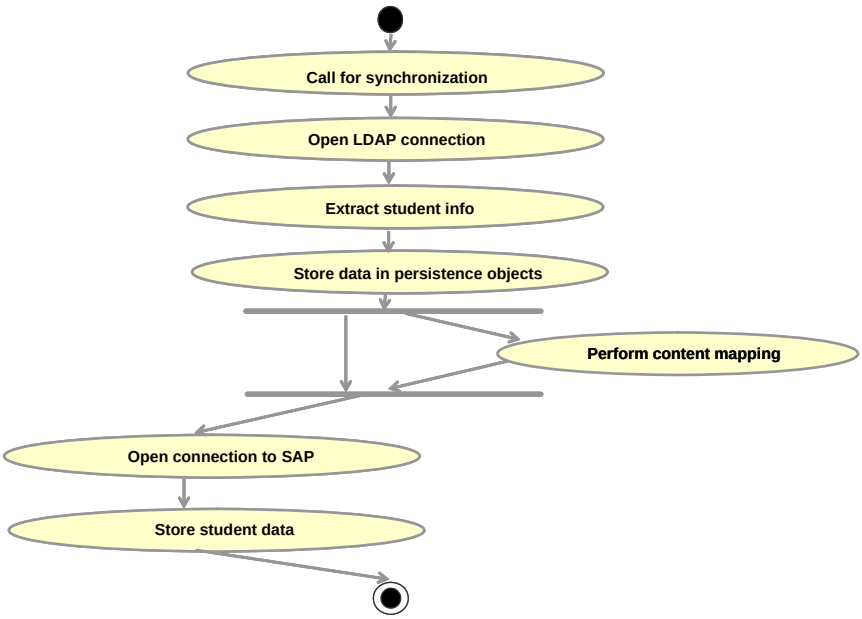
Functionally spoken, the demonstration application performs a dump of the LDAP content about students, and stores it in the Student BAPI of the SAP system, provided some minimal mapping operations. Here is a description of the use case that the demonstration application performs:

Name of Increment	Student Data Dump
Short Description	A simple dump of all available student data from the LDAP server into the SAP student BAPI
Pre-Conditions	LDAP server connection can be opened with valid credentials, SAP connection can be opened with valid mandant and credentials
Post-Conditions	The LDAP data about students in all departments is also available on the student BAPI of the SAP system
Main Flow	As according to Information 56: First Increment
Exception Flow	The provided credentials are not valid. No data will be stored in the student BAPI. Both states of the LDAP server and of the SAP BAPI stay as before the execution of the flow.



Information 56: First Increment Main Flow

Information 57: Activity Diagram for first Increment presents the execution of the Main Flow in an intuitive manner. It lets the several steps of the mini-synchronization be ordered and the interaction between the different implemented entities be shown. The steps where many arrows are gathered by an oval shape are representing internal data flows.



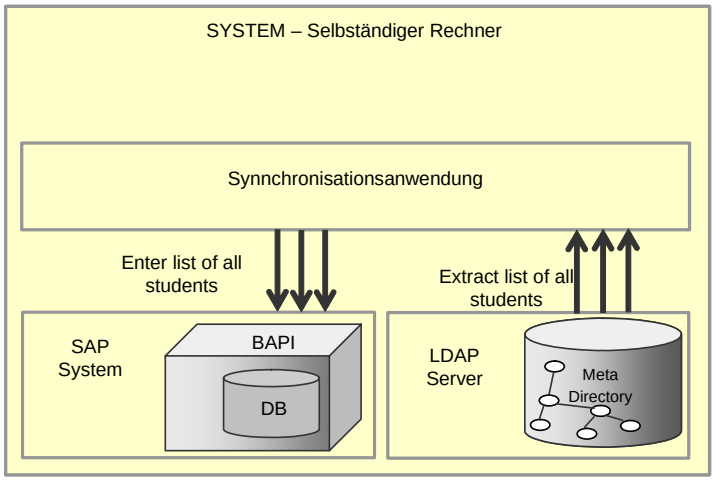
Information 57: Activity Diagram for first Increment

The upper diagram (Information 57: Activity Diagram for first Increment) supports the chart Information 56: First Increment Main Flow that describes the main flow of the implemented use case. It is a UML activity diagram, and we can see how straight-forward the execution is. It is to note that the execution of a mapping is optional since it depends on the incompatibility of data between the LDAP and the BAPI data. For example, a call to the mapper will be executed only if it is desired that some content provided from the LDAP server shall not be inserted in the SAP BAPI, or if some of the input data to the BAPI is lacking in the extracted data from the LDAP

server. As you can see, the current implementation of the synchronization process is as simple as it can get.

3.3 Deployment

3.3.1 Deployment Context



Information 58: System Implementation of Demo Application

In Information 58: System Implementation of Demo Application we present the fairly simple deployment architecture of the demonstration application. Some observations have to be made:

- Although the SAP client and the LDAP server are installed on the same machine they do not communicate directly but do it over a TCP/IP communication layer.
- The SAP system is installed on a Unix machine and acts as a server for the SAP client installed on the demonstration machine.
- There is no GUI or what so ever to execute the application, it has then to be directly called from the XP command console, by executing the “Synchronizer.java” file of the “SynchronizerCase” Package.

3.3.2 LDAP Side

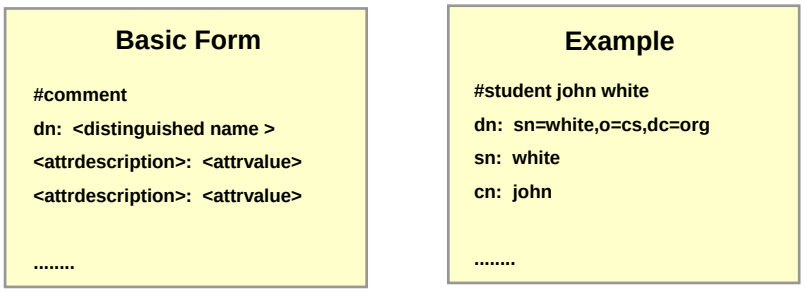
Creation of LDAP Content (LDIF)

In this section, we give a brief overview of how we created entries in LDAP. We have used LDAP Data Interchange Format (LDIF) files, in order enter content to the LDAP. LDIF is used to represent LDAP entries in a simple text format. The basic format of an entry in LDIF file is shown in Information 59: LDIF.

(1) LDAP Data Interchange Format

(2) Represent LDAP entries in a simple text format

(3) Used to build directory content in LDAP



Information 59: LDIF

The *slappadd* command in LDAP is used to add the respective content in the LDIF file to the directory. The content for organizations, organizational roles and students are added to the directory using LDIF files. The LDIF we have used to insert a student to the directory is shown in Information 60.

```
dn: deEduPersonMatrNum=125125125,ou=Student,o=Informatik,dc=uni-karlsruhe,dc=de
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: deTestEduPerson
objectclass: eduPerson
cn: Johannes Jacob
sn: Kegel
deEduPersonUserId: 125125125
deEduPersonMatrNum: 125125125
deEduPersonNationality: German
deEduPersonFieldofStudy: Informatik
deEduPersonAcademicSemester: 11
deEduPersonGender: Male
deEduPersonTitle: Herr
deEduPersonBirthDate: 1980-5-25
deEduPersonMatriDate: 2000-4-20
deEduPersonMaritalStatus: Married
mail: kegel@ira.uka.de
givenName: Johannes Jacob Friedrich
homePhone: +491762226
homePostalAddress: obere BergStr 18, 77777, Heidelberg
postalAddress: obere BergStr 18, 77777, Heidelberg
postalCode: 77777
street: obere BergStr 18
telephoneNumber: +4917878878
eduPersonAffiliation: Student
eduPersonPrimaryAffiliation: Student
eduPersonOrgDn: o=Informatik,dc=uni-karlsruhe,dc=de
eduPersonOrgUnitDn: ou=Student,o=Informatik,dc=uni-karlsruhe,dc=de
```

Information 60: Inserting a Student Entry to the Directory / LDIF

The above LDIF file shows how we have inserted a student to the directory. As you can see at the top it defines the “dn” and then the object classes it is using for this entry. There we have specified “eduPerson” and “deEduPerson”, since we are using both of them. The attributes we have specified are defined in the object classes that are defined at the top in the LDIF file.

3.3.3 SAP Side

Use of the Students' BAPI

The Java/JCO Wrapper is a class which primary represents the business object student. Any instance of the class StudentImport can be seen as a student, which has to be integrated into to the SAP system. These objects have a certain set of attributes. These attributes are equivalent to the attributes of the BAPI, which is used for the import. The Wrapper guarantees that the set of attributes are always enough and in the right data types, so that it can be imported successfully. In order to realize this, it offers some methods for setting the values of the attributes. The meaning and the list of attributes of the BAPI are described in chapter 2.

Below, you can see the available methods with a description of the class StudentImport.

Method name and attributes	**Attributes**	**Method description**
StudentImport	-	Default Constructor: The constructor does nothing except the instantiation of an object of the class StudentImport. If one call import_data after using this constructor, the import would fail.
StudentImport	firstName, lastName	Constructor with a minimum data_set. The constructor instantiates the object and sets the minimum number of attributes needed for the import.
Connect		Connection to the SAP BAPI Server. Here the server and login data of the BAPI is hard coded inside the method.
getRepository		This method is used for loading the later called remote function module.
setImportData		This method sets the import data during the BAPI-Call. It uses the attributes, which are set before the call import_data. If a needed attribute wasn't set, then this method uses default attributes.
execute		This methods executes and ends the import.
disconnect		This is used for disconnecting to the BAPI Server.

import_data		This is the global import method, which execute following 5 class internal methods.
		Connect();
		getRepository();
		setImportData();
		execute();
		disconnect();

setDefault		This method sets default attributes to the business object student. It can be used in the first development step for testing the connection to the SAP R/3 system.

It is important to mention that the SAP R/3 system has some restriction in setting your own matriculation number. Of course you can't set a number, which already exists inside the system.

And it is also not possible to set a matriculation number inside some number areas. At least it is important to mention that, if the matriculation number is not set, SAP sets it on its own.

3.4 Exercises

(1) What would be the advantages of using the JNDI API rather than NOVELL's JLDAP foundation classes to interact with the openLDAP server?

(2) Can you recall the sequence of operations executed by the synchronizer class, in the presented minimal synchronization demonstration application?

(3) What steps should an import have in general ?

(4) Why does the SAP R/3 system has the BAPI BAPI_TRANSACTION_COMMIT ?

Information 61: Exercises

(1) Information 44: JNDI Architecture

(2) Information 56: First Increment Main Flow

(3) Information 54: Process Sequences at the Import

(4) Information 54: Process Sequences at the Import

4 OUTLOOK

In this case study we primary concentrate on the import of student data from the meta directory to the SAP R/3 system. This is, because we want to introduce a basic concept how one can solve this problem in general. In the next step this one way import should be extended the other way round. That means the user should have the opportunity to export data from the SAP R/3 legacy system to meta directory. Inside this analysis one can get a lot of experience and knowledge, how one can write date from the legacy system into a LDAP meta directory. After getting good knowledge in this field, the next step would be to adapt this two synchronization functionality in a way that a complete synchronization process is possible. There has to be introduces an event mechanism in order to get the information whether something changed inside the SAP R/3 or inside the LDAP directory during very less time. Finally the whole things should be synchronized in a way that the data can be stored completely redundantly and in a way that it doesn't matter where someone integrates additional data and additional students, and the framework should synchronize this student data automatically.

APPENDIX A JAVA SOURCE CODE (JCO / LDAP)

```
/*
 * Created on 31. Januar 2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package SystemInterface;

import com.sap.mw.jco.*;
import java.lang.String;
/**
 * @author Thomas Mathes
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */


public class StudentImport
{
	public static JCO.Client	myClient;
	public static JCO.Repository	myRepository;
	public static JCO.Function	myFunction;
	public static JCO.Function	myFunction2;

	public static String		g_firstName;
	public static String		g_lastName;

	public static IFunctionTemplate	ft;
	/**
	 *
	 */
	public StudentImport ()
	{
		super();
		setDefault();
		// TODO Auto-generated constructor stub
		System.out.println("reaching for SAP system...");
	}

	public static void setDefault ()
	{
		g_lastName   =	"Default LastName";
		g_firstName  =	"First Name";
		// TODO Auto-generated constructor stub
		System.out.println("reaching for SAP system...");
	}

	public StudentImport (String firstName, String lastName)
	{
		super();
		// TODO Auto-generated constructor stub
		g_firstName  =	firstName;
		g_lastName   =	lastName;
		System.out.println("reaching for SAP system...");
	}

	public static void Connect()
	{
		myClient = JCO.createClient("223", "ABAP3", "ABAP3", "DE",
					"rzdb10.rz.uni-karlsruhe.de", "00");
		try
```

```
        {
                myClient.connect();
                System.out.println(myClient.getAttributes().getHost());
                System.out.println(myClient.getAuthorizationTraceID());
        }
        catch (Exception e)
        {
                System.out.println("An exception occured while
                                    accessing SAP System...");
                e.printStackTrace();
                System.exit(1);
        }
}

public static void getRepository()
{
        myRepository = new JCO.Repository("myRepository", myClient);

        try
        {

                Ft = myRepository.getFunctionTemplate ("
                        Z_CM_STUDENT_PROVISIONING ");

                if (ft == null)
                {
                        System.out.println("BAPI:
                                Z_CM_STUDENT_PROVISIONING nicht gefunden");
                }
                else
                {
                        System.out.println("BAPI:
                                Z_CM_STUDENT_PROVISIONING gefunden");
                        myFunction2 = ft.getFunction();
                }
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
}

public static void setImportData()
{
        JCO.ParameterListmyImport = myFunction2.getImportParameterList();
        myImport.getStructure("STAMMDATEN").setValue(g_firstName,
                "FIRST_NAME");

        myImport.getStructure("STAMMDATEN").setValue(g_lastName,
                "LAST_NAME");

        myImport.getStructure("STAMMDATEN").setValue("DE",
                "CORRESP_LANGUAGE");

        myImport.getStructure("STAMMDATEN").setValue("DE",
                "CORRESP_LANGUAGE_ISO");

        myFunction2.setImportParameterList(myImport);
}

public static void execute()
{
        myClient.execute(myFunction2);
}

public static void disconnect()
{
```

```
            myClient.disconnect();
      }

      public static void import_data()
      {
            Connect();
            getRepository();
            setImportData();
            execute();
            disconnect();
      }



/*
 * Created on 26 janv. 2006
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package SynchronizerCase;

import java.util.Iterator;
import java.util.Properties;
import java.util.Vector;

import javax.naming.NamingEnumeration;
import javax.naming.directory.Attributes;
import javax.naming.directory.BasicAttribute;
import javax.naming.directory.BasicAttributes;
import javax.naming.directory.SearchResult;

import SystemInterface.StudentImport;
import SystemInterface.*;



/**
 * @author lordmarwanus
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Synchronizer {

private DeEduPerson adep;
private Student student;
private Properties prop;
private Vector vecser;
private Vector vecstu;
private Mapper studentmapper;

      private void getStudentsFromLDAP()
      {
         LdapInterface linterface = new LdapInterfaceImpl();
         linterface.setEduPerson(new Student());
         linterface.connectToServer("localhost","cn=Manager,dc=uni-
karlsruhe,dc=de","secret", "dc=uni-karlsruhe,dc=de");
         vecstu = linterface.getEduPersonAll();
      }

     public void synchronize()
     {
          this.getStudentsFromLDAP();
    //this.studentmapper = new Mapper();
    //this.studentmapper.setStudentList(vecstu);
    //this.studentmapper.mapTypes("sn-cn");
```

```
          Iterator it = vecstu.iterator();
    while (it.hasNext())
    {
      Student tmpstudent = (Student) it.next();
StudentImport   myImport   =   new   StudentImport(tmpstudent.getGivenName(),
tmpstudent.getSurname());
myImport.import_data();
    }
}

  public Vector getStudentList(){
    return this.vecstu;
  }

public DeEduPerson getSyncDeEduPerson(){
return this.adep;
}

public void setSyncDeEduPerson(DeEduPerson adep){
this.adep = adep;
}

public Student getSyncStudent(){
return this.student;
}

public void setSyncStudent(Student student){
this.student = student;
}

public static void main(String args[])
{
System.out.println("Starting Import");
(new Synchronizer()).synchronize();
System.out.println("End Import");
}
}



/*
 * LdapInterface.java
 *
 * Created on 1. Februar 2006, 11:27
 *
 */

package SystemInterface;

import java.util.*;
/**
 * @author Praharshana
 */

/**
 * LDAP Wrapper class, provides the interface to the LDAP Directory
 */

public interface LdapInterface {

      // Establish a connection with the LDAP Server
      public void connectToServer(String ldapServerName,String rootdn,String
                                  rootpass,String rootContext);


      // Get an eduPerson object in the directory using a DN
```

```
      public Object getEduPersonWithDN(String dname);
     // Get all the objects in the directory
     public Vector getEduPersonAll();

     // specifiy which type of objects, should be retrieved
     public void setEduPerson(DeEduPerson deperson);

}



/*
 * LdapInterfaceImpl.java
 *
 * Created on 1. Februar 2006, 11:36
 *
 */

package SystemInterface;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.naming.NameAlreadyBoundException;
import javax.naming.directory.*;
import javax.naming.*;
import java.util.*;

/**
 *
 * @author Praharshana
 */
/**
 * LdapInterfaceImpl implements the LdapInterface using JNDI
 */
public class LdapInterfaceImpl implements LdapInterface {

    public DirContext dircontext;
    public DeEduPerson deEduPerson;

    public void setEduPerson(DeEduPerson deperson)
    {
      deEduPerson = deperson;
    }

    public void connectToServer(String ldapServerName,String rootdn,String
                               rootpass,String rootContext)
    {
        Properties env = new Properties();

        env.put(  Context.INITIAL_CONTEXT_FACTORY,
                 "com.sun.jndi.ldap.LdapCtxFactory" );
        env.put( Context.PROVIDER_URL,
                  "ldap://" + ldapServerName + "/" + rootContext );
        env.put( Context.SECURITY_PRINCIPAL, rootdn );
        env.put( Context.SECURITY_CREDENTIALS, rootpass );

        try {
          dircontext = new InitialDirContext(env);
        } catch ( NameAlreadyBoundException nabe ) {
        System.err.println( "value has already been bound!" );
        } catch ( Exception e ) {
        System.err.println( e );
        }
    }

    private Object getEduPerson(Attributes attributes)
```

```
    {
        return deEduPerson.getEduPersonFromLDAPServer(attributes);
    }

    public Object getEduPersonWithDN(String dname)
    {
        return null;
    }


    public Vector getEduPersonAll()
    {
        Vector students = new Vector();
        SearchControls ctls = new SearchControls();
        ctls.setSearchScope(SearchControls.SUBTREE_SCOPE);
        String filter = "(deEduPersonMatrNum=*)";
        try
        {
                NamingEnumeration answer = dircontext.search("",
                                                        filter, ctls);
                while (answer.hasMore())
                {

                          SearchResult sr   = (SearchResult)answer.next();
                          Attributes attr   = sr.getAttributes();
                          Student student   = (Student)getEduPerson(attr);
                          students.add(student);
                }
        }catch(javax.naming.NamingException ne)
        {
            System.out.println(ne.toString());
        }
        return students;
    }


}



/*
 * DeEduPersonImpl.java
 *
 * Created on 22. Januar 2006, 22:22
 *
 */

package SystemInterface;

import javax.naming.directory.*;
/**
 *
 * @author Praharshana
 */
/**
 * Interface DeEduPerson for the objects in LDAP
 */
public interface DeEduPerson {


       // Retrieves a specific type of object in the directory
      public Object getEduPersonFromLDAPServer(Attributes attributes);


}
```

```
/*
 * Student.java
 *
 * Created on 22. Januar 2006, 22:31
 *
 */

package SystemInterface;

import javax.naming.directory.*;
/**
 *
 * @author Praharshana
 */
/**
 * Class Student, which represent the Student objects
 * in LDAP and implements DeEduPeron interface
 */
public class Student implements DeEduPerson
{
    private String deEduPersonUid;
    private String eduPersonOrgUnitDN;
    private String homePhone;
    private String givenName;
    private String deEduPersonNationality;
    private String deEduPersonBirthDate;
    private String postalCode;
    private String eduPersonAffiliation;
    private String deEduPersonGender;
    private String mail;
    private String cn;
    private String telephoneNumber;
    private String street;
    private String deEduPersonMatrNum;
    private String deEduPersonMatriDate;
    private String postalAddress;
    private String surname;
    private String deEduPersonTitle;
    private String deEduPersonMaritalStatus;
    private String homePostalAddress;
    private String deEduPersonAcademicSemester;
    private String eduPersonPrimaryAffiliation;
    private String eduPersonOrgDN;
    private String deEduPersonFieldofStudy;

    public String getUid()
    {
        return deEduPersonUid;
    }

    public String getEduPersonOrgUnitDN()
    {
        return eduPersonOrgUnitDN;
    }

    public String getHomePhone()
    {
        return homePhone;
    }

    public String getGivenName()
    {
       return givenName;
    }
```

```
 public String getNationality()
 {
    return deEduPersonNationality;
}

public String getBirthDate()
{
   return deEduPersonBirthDate;
}

public String getPostalCode()
{
    return postalCode;
}

public String getEduPersonAffiliation()
{
    return eduPersonAffiliation;
}

public String getGender()
{
    return deEduPersonGender;
}

public String getEmail()
{
    return mail;
}

public String getCommonName()
{
    return cn;
}

public String getTelephoneNumber()
{
    return telephoneNumber;
}

public String getStreet()
{
    return street;
}

public String getMatriculationNumber()
{
    return deEduPersonMatrNum;
}

public String getMatriculatedDate()
{
    return deEduPersonMatriDate;
}

public String getPostalAdress()
{
    return postalAddress;
}

public String getSurname()
{
    return surname;
}

public String getTitle()
{
```

```
    return deEduPersonTitle;
}

public String getHomePostalAdress()
{
    return homePostalAddress;
}

public String getAcademicSemester()
{
    return deEduPersonAcademicSemester;
}

public String getFieldOfStudy()
{
    return deEduPersonFieldofStudy;
}

public String getMaritalStatus()
{
   return deEduPersonMaritalStatus;
}


 public Object getEduPersonFromLDAPServer(Attributes attributes)
 {

     Student student = new Student();
     try
     {
      student.eduPersonOrgUnitDN
      =  attributes.get("eduPersonOrgUnitDN").get().toString();
      student.homePhone
      = attributes.get("homePhone").get().toString();
      student.givenName
      = attributes.get("givenName").get().toString();
      student.deEduPersonNationality
      = attributes.get("deEduPersonNationality").get().toString();
      student.deEduPersonBirthDate
      = attributes.get("deEduPersonBirthDate").get().toString();
      student.postalCode
      = attributes.get("postalCode").get().toString();
      student.eduPersonAffiliation
      = attributes.get("eduPersonAffiliation").get().toString();
      student.deEduPersonGender
      = attributes.get("deEduPersonGender").get().toString();
      student.mail
      = attributes.get("mail").get().toString();
      student.cn
      = attributes.get("cn").get().toString();
      student.telephoneNumber
      = attributes.get("telephoneNumber").get().toString();
      student.street
      = attributes.get("street").get().toString();
      student.deEduPersonMatrNum
      = attributes.get("deEduPersonMatrNum").get().toString();
      student.deEduPersonMatriDate
      = attributes.get("deEduPersonMatriDate").get().toString();
      student.postalAddress
      = attributes.get("postalAddress").get().toString();
      student.surname
      = attributes.get("sn").get().toString();
      student.deEduPersonTitle
      = attributes.get("deEduPersonTitle").get().toString();
      student.deEduPersonMaritalStatus
      =  attributes.get("deEduPersonMaritalStatus").get().toString();
      student.homePostalAddress
```

```
          = attributes.get("homePostalAddress").get().toString();

        student.deEduPersonAcademicSemester
        = attributes.get("deEduPersonAcademicSemester").get().toString();
        student.eduPersonPrimaryAffiliation
        = attributes.get("eduPersonPrimaryAffiliation").get().toString();
        student.eduPersonOrgDN
        =   attributes.get("eduPersonOrgDN").get().toString();
        student.deEduPersonFieldofStudy
        = attributes.get("deEduPersonFieldofStudy").get().toString();


        }
         catch(javax.naming.NamingException e)
              {
              }

        return student;
    }

}
```

APPENDIX B BAPI INTERFACE DESCRIPTION

```
FUNCTION Z_CM_STUDENT_PROVISIONING.
*"----------------------------------------------------------------------
*"*"Lokale Schnittstelle:
*"  IMPORTING
*"     VALUE(MATRNR) TYPE  BAPISTUDENT_HEAD-STUDENTNUMBER OPTIONAL
*"     VALUE(STAMMDATEN) TYPE  BAPISTUDENT_PERSONAL
*"  EXPORTING
*"     VALUE(RETWERT) TYPE  BAPIRET2_T
*"----------------------------------------------------------------------

DATA:
  retu type bapiret2_t.

*---- Aufrufen des BAPI zur Ermitteln der Studentenstammdaten
    CALL FUNCTION 'BAPI_STUDENT_CREATEFROMDATA3'
         EXPORTING
              STUDENTPERSONALDATA = STAMMDATEN
              STUDENTNUMBEREXTERN = MATRNR
         TABLES
              RETURN = retu.

*---- Aufrufen des BAPI zur Ermitteln der Studentenadressdaten
    CALL FUNCTION 'BAPI_TRANSACTION_COMMIT'.

    RETWERT = retu.

ENDFUNCTION.
```

APPENDIX C LDAP PLATFORM INSTALLATION AND CONFIGURATION

Many commercial solutions are available to build an LDAP directory. However most products are either very expensive or not standard compliant. Therefore we are using OpenLDAP, an opensource LDAP server designed for UNIX platforms. Luckily we could find a Windows release of OpenLDAP because our system is running on the Windows environment. This Windows release of OpenLDAP is available at http://lucas.bergmans.us/hacks/openldap/. In the next section we will give a brief introduction of the installation and configuration of OpenLDAP for Windows.

OpenLDAP Installation

The OpenLDAP installer for windows can be downloaded from: http://download.bergmans.us/openldap. Launch the installer, accept license and you must choose target installation directory, here we have used the default directory: c:\openldap.

The installer automatically builds a valid configuration. The OpenLDAP startup files are stored in c:\openldap\var and the data files are in c:\openldap\var\openldap-data.

OpenLDAP Configuration

Before launching the LDAP server we have to customize it for our organizational requirements. The main LDAP configuration file is slapd.conf in the default directory c:\openldap. We have to update this file according to our requirements. The changes to be done to the slapd.conf file are listed below.
An example configuration file, what we have specified in our configuration is shown below.

```
ucdata-path     C:/openldap/ucdata
include         C:/openldap/etc/schema/core.schema
pidfile         C:/openldap/var/slapd.pid
argsfile        C:/openldap/var/slapd.args
database        bdb
suffix          "dc=uni-karlsruhe,dc=de"
rootdn          "cn=Administrator,dc=uni-
karlsruhe,dc=de"
rootpw          ourpassword
directory       C:/openldap/var/openldap-data
index           objectClass          eq
```

Information 62: Configuration File Example

1. Specify the Unicode data directory, by default c:\openldap\ucdata.
2. Choose the needed LDAP schemas. The core schema is mandatory but here we have to specify the schemas we use, for example eduPerson
3. Configure the path for OpenLDAP pid and args startup files. The first contains the server pid, the second includes command line arguments
4. Choose the database type, by default bdb (Berkeley DB).
5. Specify the server suffix. All entries in the directory will have this suffix, which represents the root of the directory tree. For example, with suffix "dc=uni-karlsruhe,dc=de", the fully qualified name of all entries in the database will end with : dc=uni-karlsruhe,dc=de.
6. Define the name of the administrator entry for the server, named the rootdn, along with its password rootpw. This is the super user of the server. The rootdn name must match the

suffix defined above. As stated, all entry names must end with the suffix, and the rootdn is an entry.

Using your LDAP server

Start the server

To start the LDAP server you can either double click slapd.exe executable in c:\openldap or launch it from the command line. Here you won't get any messages but you must leave the command window open for further steps. If you want debug information, you can use the –d option. In the command like this command will look like

```
C:>slapd –d 1
```

Testing your server

The OpenLDAP command line tools can be used to test the server. The following command executes a search query on the server:

```
ldapsearch -x -s base (objectclass=*) namingContexts
```

Type this command in a windows command window positioned in the OpenLDAP installation directory. For more information on search syntax, try:

```
ldapsearch --help.
```

Insert some Content in Meta Directory:

In order to insert the first entries in the database, create an init.ldif file and add the following content (according to the configuration specified in slapd.conf):

```
dn: dc=uni-karlsruhe,dc=de
objectclass: top
objectclass: dcObject
objectclass: organization
o: uni-karlsruhe
dc: de
dn: cn=Manager,dc=uni-karlsruhe,dc=de
objectclass: organizationalRole cn: Manager
```

Information 63: Initial LDIF file (init.ldif)

Create this file directly in OpenLDAP installation directory or anywhere after adding the installation directory to the system PATH.
Load entries in directory (adjust slapd.conf and init.ldif files path as needed):

```
slapadd -f slapd.conf -l init.ldif
```

Install your LDAP service:

In order to have the server always available, register OpenLDAP as a Windows service. To achieve this, just use the simple command:

```
slapd install
```

And to remove the service:

```
slapd remove
```

OpenLDAP daemon parameters can be modified by creating registry keys. Create a .reg file with the following content en register keys by double clicking on it:

```
REGEDIT4 // Muss ANGEPASST werden :: PRAHARSHANA
[HKEY_LOCAL_MACHINE\SOFTWARE\OpenLDAP]
@="c:\\openldap"
[HKEY_LOCAL_MACHINE\SOFTWARE\OpenLDAP\Parameters]
"DebugLevel"=dword:00000000
"ConfigFile"=".\\slapd.conf"
"Urls"="ldap:///"
```

Information 64: OpenLDAP Daemon Parameters Modification

The first key contains OpenLDAP installation path. DebugLevel is used to change trace level, ConfigFile is the configuration file path and URLs matches the syntax of the -h command line option.
For example, set URLs to ldap://localhost:port/ to set a different listen port.

For more detailed information about this installation process, you may find this URL helpful: http://mguessan.free.fr/nt/openldap_en.html.

TABLES

Abbreviations and Glossary

Abbreviation or Term	Full Name and/or Term Description
ABAP	Advanced Business Application Programming Programming language for the development of applications for SAP systems.
BAPI	Business Application Programmer Interface It is a special interface of SAP R/3, which represents a business object and allows the application developer to import or export this business object from and into the SAP R/3 legacy system.
BPMN	Business Process Modeling Notation Important specification in the context of Business Process Modeling (BPM) developed by the Business Process Management Initiative (BPMI).
DIT	Directory Information Tree The LDAP naming model defines how entries are identified and organized. Entries are organized in a tree-like structure called the Directory Information Tree. Entries are arranged within the DIT based on their distinguished name
EAI	Enterprise Application Integration EAI (Enterprise Application Integration) is a business computing term for the plans, methods, and tools aimed at modernizing, consolidating, and coordinating the computer applications in an enterprise. Typically, an enterprise has existing legacy applications and databases and wants to continue to use them while adding or migrating to a new set of applications that exploit the Internet, e-commerce, extranet, and other new technologies. EAI may involve developing a new total view of an enterprise's business and its applications, seeing how existing applications fit into the new view, and then devising ways to efficiently reuse what already exists while adding new applications and data.
eduPerson	An LDAP object class authored and promoted by the EDUCAUSE/Internet2 eduPerson Task Force to facilitate the development of inter-institutional applications. The eduPerson object class focuses on the attributes of individuals.
Identity Management (IdM)	Identity management involves all aspects of managing users (digital entities) in the enterprise application environment. This includes how users are created, how they are granted access privileges, how their access to applications is controlled and managed, and how these events are tracked and reported.
JCo	SAP Java Connector A Java-based middleware, acting as a bridge between ABAP and Java.

LDAP	Lightweight Directory Access Protocol LDAP is designed to provide access to directories supporting the X.500 models, while not incurring the resource requirements of the X.500 Directory Access Protocol (DAP). This protocol is specifically targeted at management applications and browser applications that provide read/write interactive access to directories. When used with a directory supporting the X.500 protocols, it is intended to be a complement to the X.500 DAP [IETF-RFC2251].
LDAP Schema	An LDAP schema defines a set or rules that specifies the types of objects that a directory may contain and the required and optional attributes that entries of different types should have. It may also specify the structure of the namespace and the relationship between different types of objects.
Legacy System	A legacy system is an antiquated computer system or application program which continues to be used because the user (typically an organization) does not want to replace or redesign it.
Meta directory	A meta directory is the primary operational repository used by a meta directory service [BG-GLOS].
Wrapper	A Wrapper is a library which hides a complex interface to a system and offers a simple and scenario spcific interface to the application developer.

Index

ABAP 35
BAPI 21
BPMN 10
Directory Information Tree 31
EAI 6
eduPerson 15
Identity Management (IdM) 6
JCO 34
LDAP 12
LDAP Schema 14
Legacy System 9
Meta Directory 9
Wrapper 52

Information and Exercise Slides

Information 1: Introduction 6
Information 2: Study Progress 7
Information 3: Decentralized Synchronization of the Student data 8
Information 4: Scenario of the Synchronization Process 9
Information 5: Main Concerns of Synchronization Process 9
Information 6: Process Modeling Decisions 10
Information 7: Design of the Synchronization Process 10
Information 8: Business Object Diagram 11
Information 9: Introduction to Directories 12
Information 10: Introduction to LDAP 13
Information 11: LDAP Schema 13
Information 12: Attributes 14
Information 13: Object Classes 15
Information 14: eduPerson Schema 16
Information 15: Attributes in eduPerson 17
Information 16: Overview of the DEEP Questionnaire 18
Information 17: Location of the Legacy System in our Scenario 19
Information 18: 3-Level Architecture of SAP R/3 20

Information 19: Different SAP modules 21
Information 20: Access to the Business Object Student over BAPI 21
Information 21: Overview of the Student Data Structure of the BAPI 22
Information 22: Ruled Synchronization & Event Registration 23
Information 23: Exercises 24
Information 24: Global Application Architecture 26
Information 25: Integration Layers Diagram 27
Information 26: Schema Design 28
Information 27: Attributes in deEduPerson 29
Information 28: DIT for Student Objects in the Directory 31
Information 29: LDAP Communication with an Application 32
Information 30: LDAP Communication with a Java Application 33
Information 31: Application Accessing Student Data with DN 33
Information 32: Physical Separation of SAP and Middleware (Java/JCO) 34
Information 33: Data flow between Java and SAP 34
Information 34: Technical Access Interface between JCO and BAPI 35
Information 35: SAP Internal Structure of BAPIs, RFC and ABAP 35
Information 36: Design Phase – Synchronization: Data Access 36
Information 37: Refined Synchronization Components Interfacing Model 37
Information 38: Packages Diagram 37
Information 39: Synchronizer Classes 38
Information 40: Data Format Rules 39
Information 41: System Interfaces 39
Information 42: Business Rules 39
Information 43: Exercises 40
Information 44: JNDI Architecture 43
Information 45: Packages of JNDI 44
Information 46: Naming Service Example 45
Information 47: JNDI Wrapper 47
Information 48: Connecting to LDAP Server 48
Information 49: Searching the Directory 49
Information 50: The Synchronizer Class Implementation 50
Information 51: The LDAP Export Wrapper Class Implementation 51
Information 52: The SAP Import Wrapper Class Implementation 51
Information 53: Advantages of a Java/JCO – Wrapper 52
Information 54: Process Sequences at the Import 54
Information 55: JCO Implementation, Access to SAP R/3 55
Information 56: First Increment Main Flow 56
Information 57: Activity Diagram for first Increment 56
Information 58: System Implementation of Demo Application 57
Information 59: LDIF 58
Information 60: Inserting a Student Entry to the Directory / LDIF 58
Information 61: Exercises 60
Information 62: Configuration File Example 73
Information 63: Initial LDIF file (init.ldif) 74
Information 64: OpenLDAP Daemon Parameters Modification 75

References

[AE+04] Sebastian Abeck, Christian Emig, Jochen Weisser: Fallstudie Transcript of Records, Bericht zum Projekt „Werkstatt Unternehmenssoftware Karlsruhe“ (WUSKAR), Karlsruhe 2004.

[BG-GLOS] Burton Group: Concepts and Definitions (Glossary), Version 2.0, September 2005.

[BPI-BPMN1.0] Business Process Management Initiative (BPMI): Business Process Modeling Notation (BPMN), Version 1.0, BPMI.org, May 2004.

[C&M-ABAP] Cooperation & Management, Tomas Stiller: Einführung in ABAP, Dokument im Rahmen der C&M Technologien und Werkzeuge, Universität Karlsruhe (TH), C&M (Prof. Abeck), Mai 2005.

[C&M-I-ID] Cooperation & Management: IDENTITY MANAGEMENT IN THE FOCUS OF APPLICATION INTEGRATION, Course Unit of the Lecture INTERNET SYSTEMS AND WEB APPLICATIONS (ISWA), http://www.cm-tm.uka.de/iswa, Universität Karlsruhe (TH), C&M (Prof. Abeck).

[C&M-JCo] Cooperation & Management, Heiko Schandua: Einführung zum SAP Java Connector, Dokument im Rahmen der C&M Technologien und Werkzeuge, Universität Karlsruhe (TH), C&M (Prof. Abeck), Mai 2005.

[DEEP-A2] DAASI International Ltd, Deliverable A.2 of the Project "Definition of an European EduPerson" (DEEP), October 2002.

[EDUPERSON01] Official Homepage of Internet2/Educause eduPerson Working Group http://www.educause.edu/eduperson/

[OPENLDAP-2.3] OpenLDAP Software 2.3 Administrator's Guide http://www.openldap.org/doc/admin23/

[OPENLDAP1.0] OpenLDAP for Win32 Documentation/FAQ http://lucas.bergmans.us/hacks/openldap/doc

[S-SCHEMADES] Skills 1st LTD, Andrew Findlay: LDAP Schema Design, February 2005.

[S-JNDI] http://java.sun.com/products/jndi/tutorial/getStarted/overview/index.html

[SS05] Heiko Schandua, Tomas Stiller: Werkstatt Unternehmenssoftware Karlsruhe (WUSKAR), Case Study University SOA, Team Study Thesis, University of Karlsruhe (TH), C&M (Prof. Abeck), 2005.