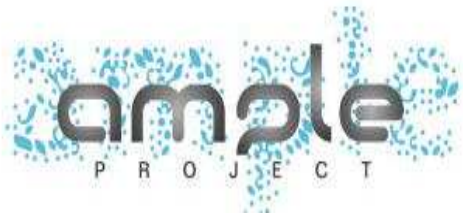

AMPLE
Aspect –Oriented, Model-Driven, Product Line Engineering
Specific Targeted Research Project: IST- 33710

Report describing existing software systems development and product line engineering practices at industrial partners

ABSTRACT
Summarizes the findings of the analysis conducted in Task 6.1 at HOLOS, SAP, and Siemens with respect to currently employed methods and techniques for implementing software product lines in productive environments.

Document ID:	AMPLE D6.3
Deliverable/ Milestone No:	D6.3
Work-package No:	WP6
Type:	Deliverable
Dissemination:	CO
Status:	Final
Version:	0.14
Date:	2007-09-22
Author(s):	Pimentão, João Paulo, (Holos), Pohl, Christoph; Rummler, Andreas (SAP); Schwanninger, Christa (Siemens)

Project Start Date: 01 October 2006, Duration: 3 years

History of Changes

Version	Date	Changes
0.1	2007-01-22	Document creation
0.2	2007-02-07	Description of approaches at SAP
0.3	2007-02-08	Illustration of SAP approaches added
0.4	2007-03-15	Description of approaches at Siemens
0.5	2007-04-03	Contribution of practices at Holos added
0.6	2007-08-13	Merge from D3.1
0.7	2007-08-20	added section about processes at SAP
0.8 – 0.10	n/a	missing, because of wrong numbering scheme
0.11	2007-09-07	added SAP contributions to section 3.1/3.2
0.12	2007-09-10	added contributions from Siemens to section 3.2
0.13	2007-09-14	version for internal review
0.14	2007-09-22	corrected some spelling mistakes, revised Holos section

Table of Contents

1. Introduction

Industrial exploitation has been institutionalised in the AMPLE work plan in the form of three consecutive tasks, 6.1 through 6.3, which also bear a strong relation to work package 5. The goal of this setup is a close alignment of concepts developed in AMPLE with industrial practice and shortcomings of currently deployed techniques.
The document at hand, D6.3, summarises the findings of task 6.1. This task's purpose was to analyse existing software systems development and software product line engineering processes and practices at industrial partners with regards to evolution to AMPLE concepts. The approach further entails using these results as input for both, WP5 and task 6.2, i.e., conveying the industrial requirements in the representative case studies and experimenting with the case studies to understand the impact on existing processes. This will ultimately lead to a generalized software process improvement framework for evolving existing processes to incorporate the AMPLE concepts, which is to be investigated in task 6.3.
The three industrial partners of AMPLE represent very different categories of industrial software development. HOLOS is an SME primarily focussed on similar custom development projects in a specific domain. SAP is a large enterprise with a homogeneous but proprietary development infrastructure for developing a family of related products. Siemens is large group of heterogeneous enterprises with whole range of existing development processes and practices. Consequentially, it is not trivial to compare their existing practices in a homogeneous way. Our approach is instead to benefit from this diversity to cover a broad range of viewpoints.
This document is structured as follows: Chapter 2 outlines the existing practices of the individual industry partners and describes potential exploitation paths. Chapter 3 summarises these findings and derives challenges for potential improvements by AMPLE concepts as an input to the technical work packages 1-4. Finally, chapter 5 concludes with an outlook on next steps.

2. Existing practices at industrial partners

The purpose of this chapter is a simple decoupled survey of existing software development and software product line engineering processes and practices at individual industry partners of AMPLE. Since these practices might not even be known internally under the term "software product line", the focus of this survey is generally on mechanisms for managing variability and commonalities of related software products in various domains.

2.1 HOLOS

Besides specific customer driven software development, HOLOS has been developing software for the European Space Agency using the Agile Modelling Methodology (http://www.agilemodeling.com).
This methodology is strongly used in projects where variability is not only a requirement at the end of the project (reusing project modules and lessons learnt from one project to another are current practice within ESA projects), but also during the development of the projects themselves.
Motivation for the use of this methodology has its roots in the need to strengthen the end-user involvement in the project and ensure the compliance with requirements throughout the development cycles. The active involvement and cooperation of the

end-users is expected and necessary to take advantage of the proposed development approach.
The assumption of the Agile Methodology requires the participation of users in the development process and the consciousness, right from the very beginning, that this involvement is necessary. Therefore, meetings (mostly teleconferences, in order to reduce costs) are organized and flexible on-line communication mechanisms implemented.

2.1.1 The Methodology

The Agile methodology envisages three major iterations for the implementation process:

a) Functional Model iteration;
b) Design & Build iteration;
c) Implementation.

Functional Model iteration

The "Functional Model iteration" identifies the user requirements and creates a mock-up of the interface that allows the user to understand how his / her requirements will be met and translated into system functionality.

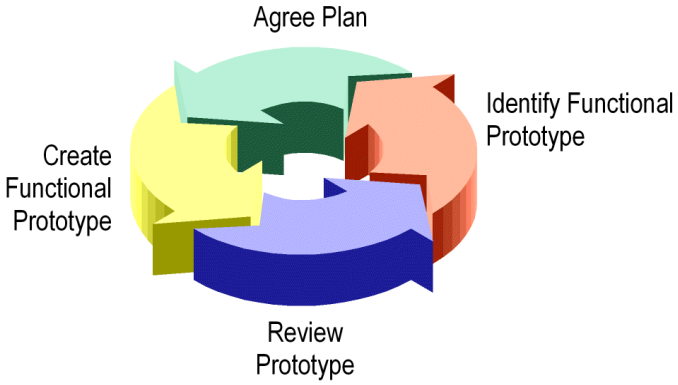

Figure 1: Functional Model iteration

The functional prototype produced at this stage is subdivided into four tasks (Figure 1):

a1) identification of the user requirements;
a2) a case-dependent definition of a specific, suitable methodology in order to effectively focus the functional prototyping and the hereby supported solution;
a3) the effective creation of the functional prototype. The goal is the construction of all necessary high level analysis models and documentation supported by functional prototypes which address detailed process and usability;
a4) the prototype review task, which is animated by the functional test of the developed prototypes.

2.1.1.1 Design & Build iteration

The design prototype implements parts of the system architecture, allowing the user to test the functionalities with real data in a controlled environment.

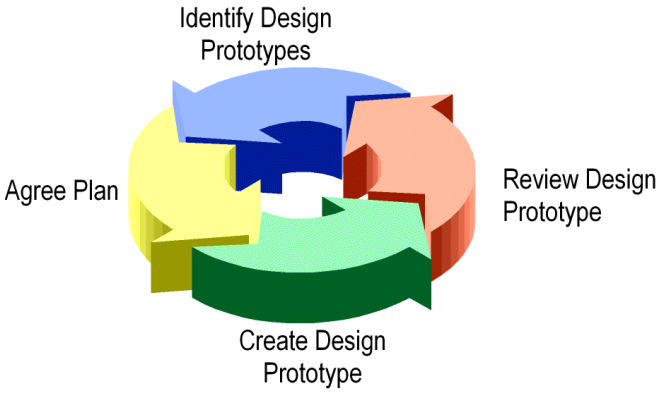

Figure 2: Design and Build iteration

It is sub-divided in the following tasks (Figure 2):

b1) identification of the Design Prototypes;

b2) as part of the joint development, the "agree plan" activity results in the most efficient approach to the design of the pieces that will constitute the target application;

b3) the design-prototype is created and related deliverables are supported by a set of tools.

b4) the review of the design prototype is based on testing against realistic data (preferably real data retrieved from the sources that will be used by the final application).

2.1.1.2 Implementation

The Implementation phase is the scenario where the latest increments in the iterative development methodology drive to a prototype that is fully released to the end user. This represents the transition from the development to the operational scenario – including final tuning – as well as the effective handing over to the end user, who – conveniently assisted - will perform the operational validation.

Figure 3: Implementation

The Implementation is subdivided into four major tasks (Figure 3):

c1) the "user guidelines" tasks provides the straightforward connection with the end user and helps to plan the last cycle of iterations;

c2) the effective implementation of the operational ready prototype;

c3) the assistance to the users in their operation in the prototype. The Handing Over Process is expected to have taken place in the meantime.

c4) Prototype presentation, demonstration and effective validation – at the client's premises – closes the nominal iteration cycle of this phase.

Further details of the methodology are available at the Agile Modeling Home Page at (http://www.agilemodeling.com/)

HOLOS view on the application of this methodology is as follows.
The results of the application of this methodology ensure that the components of software developed are fully compliant with the end-users' requirements, since they are involved throughout the whole process.
Partial test of the prototypes being developed also presents the end-user with possible limitations of the technology at an early stage, which, in turn, gives rise to revision of requirements, but also ensures that at the end, the user is presented with a system whose "usability" is directly what he/she expects.
At an early stage the prototypes are released to the end user where tests are conducted, most of the times with real data and on real operational conditions.
Early test of prototypes helps identifying possible bottlenecks (e.g. performance) and provides a forum for the discussion and selection of alternatives that effectively meet the requirements or the revision of requirements (even those introduced during the process).
The development of the prototypes is always done in a modular fashion where module interfaces are agreed upfront, thus allowing for reusability.

2.2 SAP

At SAP various techniques are already applied to handle variability in software products. This section describes technologies used in SAP ERP and its follow-up in more detail. In addition technologies and processes used during the development are also examined in more detail.

2.2.1 SAP NetWeaver Platform

SAP NetWeaver is the underlying technology platform of all SAP applications. The following figure gives an overview of the SAP NetWeaver solution map:

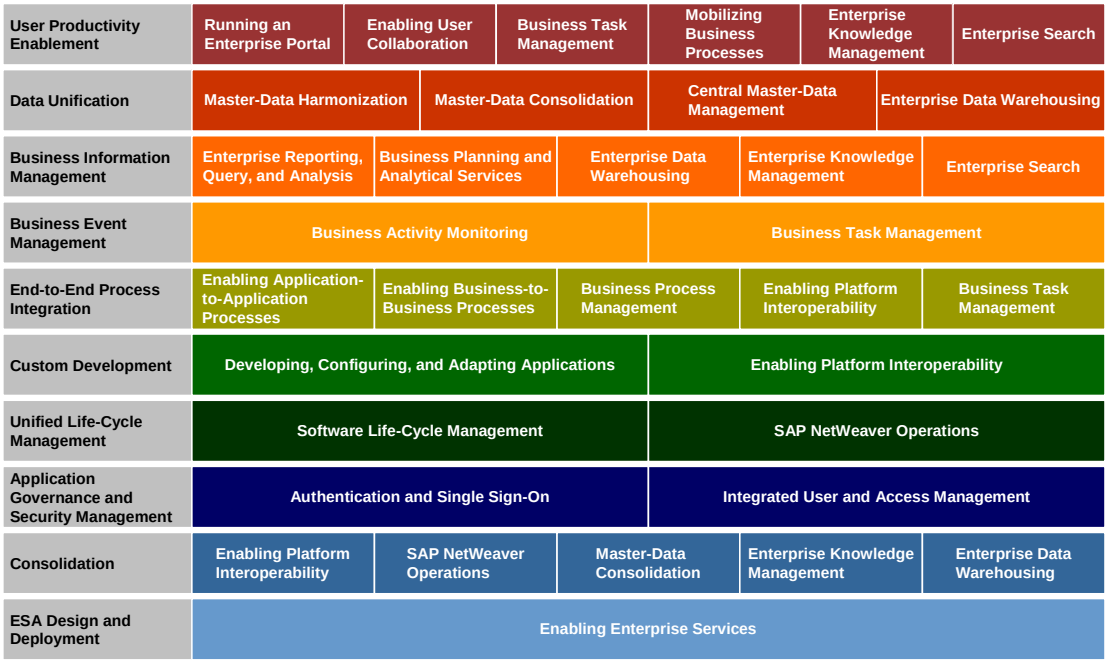

Figure 4. SAP NetWeaver Solution Map

Obviously, a whole range of different technologies, frameworks and libraries are integrated in this platform. At the bottom there are two different language stacks

At the bottom there are two different language stacks that are coexisting. SAP software may be implemented on top of both of these stacks:

1. ABAP (Advanced Business Application Programming) [Kel07] was developed and extended by SAP as the primary language for writing business applications. The ABAP stack will remain the strategic platform for business logic running on backend servers, also in the advent of the upcoming Enterprise SOA based, component-oriented Business Process Platform. Although legacy plain ABAP programs are still supported, new applications are almost exclusively written in ABAP Objects, the downward compatible Object-Oriented extension of ABAP. ABAP Objects has all major features of modern OO languages, except for method overloading. However, the lack of this feature can be circumvented by a number of alternative best practices. ABAP furthermore has a number of built-in language features like direct access to database tables, which predestine it for implementing data-intensive business software. In addition ABAP features aspect-oriented characteristics, which are explained in section 2.2.5 in greater detail.
2. Java on the other hand is primarily used for most web-based UI/portal technologies (on a JEE basis). Java also plays an increasing role for implementing service consumption and service composition on top of the Business Process Platform. This strategic decision for a wide-spread industry standard language enables SAP partners and ISVs to recruit developers from a far larger community than in a pure ABAP-based environment.

While ABAP development (programming, debugging, deployment etc.) is supported by a set of dedicated development transactions, which are executed on the host server, all Java development at SAP is performed using the client-side IDE of NetWeaver Developer Studio (NWDS). NWDS is an extension of the popular Eclipse tool platform [Cla06], [MA05], which consists of a large number of SAP-specific plug-ins. Plug-ins are the primary extension mechanism for addition new features to the Eclipse platform.

Connectivity between distributed components is established via three technologies: Remote Function Calls (RFC), the J2EE Connector Architecture (JCA) and the Java Message Service (JMS).

RFC is the standard SAP interface to communicate with SAP backend systems and non-SAP systems, where functions can be called to be executed on remote systems.

The JCA is a specification that defines the standard architecture for connecting the Enterprise Edition of the Java Platform (J2EE) to heterogeneous Enterprise Information Systems (EIS), which may include ERP and database systems. The mechanisms that the connector architecture defines are scalable and secure and enable integration of the EIS with application servers and enterprise applications. An EIS may supply so-called resource adapters, which are used to connect to the EIS. The connectors can be plugged into an application server and provide connectivity between the EIS, the application server and the enterprise application. When an application server supports this connector architecture, it provides seamless connectivity to multiple EISs.

JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise messaging product. A JMS application is made up of a set of application defined messages and a set of clients that exchange them. Products that implement JMS do this by supplying a provider that implements the JMS interfaces. Messages are asynchronous requests, reports or events that are consumed by enterprise applications.

Enterprise systems need to persist large amounts of data. To achieve this task the NetWeaver Platform enables the use of several technologies for establishing persistence.

OpenSQL is the SAP database abstraction layer implemented in ABAP that translates abstract SQL statements to native database SQL statements. OpenSQL covers the Data Manipulation Language (DML) part of the SQL standard and extends the SQL standard by offering options to simplify and accelerate database access.

Java Database Connectivity (JDBC) technology provides cross-DBMS connectivity to a wide range of SQL databases and access to other tabular data sources, such as spreadsheets or flat files. It is supported by the NetWeaver Platform for J2EE development. With a JDBC technology-enabled driver it is possible to connect all corporate data independent from homogeneous or heterogeneous environments.

The Java Data Objects (JDO) API is a standard interface-based Java model abstraction of persistence. It is supported by the NetWeaver Platform as an alternative to JDBC. JDO technology has the advantage to be able to store Java domain model instances directly in a database. The process of mapping data to relational databases is transparent for a developer.

For implementing business logic both of the language stacks mentioned above can be used. ABAP is tailored to implementing business applications. It allows quick development of business applications providing powerful macros to create the actual business logic based on SAP backend systems. There is a huge amount of existing business objects on which a developer may rely on.

The Composite Application Framework (CAF) offers a methodology and toolset to create and manage composite applications. It leverages information and data from existing applications to solutions by composing existing or new services, user interface components, and business processes. CAF is based on the Enterprise Services Architecture (ESA) and comprises an abstraction layer for services and processes as well as design tools and integrates many key capabilities of the NetWeaver Platform.

In the area of user interaction Web Dynpro is the recommended NetWeaver programming model. The Web Dynpro model is based on the Model-View-Controller (MVC) programming model and allows a clear separation of business logic and display logic. The development environment provides powerful graphical tools to layout the user interface.

However, there are other technologies that are supported alongside. Business Server Pages (BSP) are a page-based Web programming model with server-side scripting in ABAP. BSPs gives complete freedom when designing UIs since any HTML and/or JavaScript can be sent to the client. With the HTMLB BSP extension SAP also offers a library of predefined UI elements that simplify the creation of BSP pages. The pendant are Java Server Pages which enable page-based web programming with server-side scripting in Java. In addition there are frameworks on a higher abstraction

level like for instance Guided Procedures (GP). GP provides tools and a framework for modelling and executing user-oriented workflows. It supports business specialists in implementing processes and guides casual users through the execution of these processes.

2.2.2 SAP Exchange Infrastructure (XI)

An important cornerstone of integration technology built into the NW platform is the SAP Exchange Infrastructure (XI) [Stu05], an Enterprise Application Integration (EAI) solution supporting also message-oriented / event-driven "hub and spoke" [Bus03] style business-to-business (B2B) interactions, which loosely couple heterogeneous applications. This corresponds to event-based component interaction as a highly modular architecture style with independent structures whose variability can be bound very late in software lifecycle.

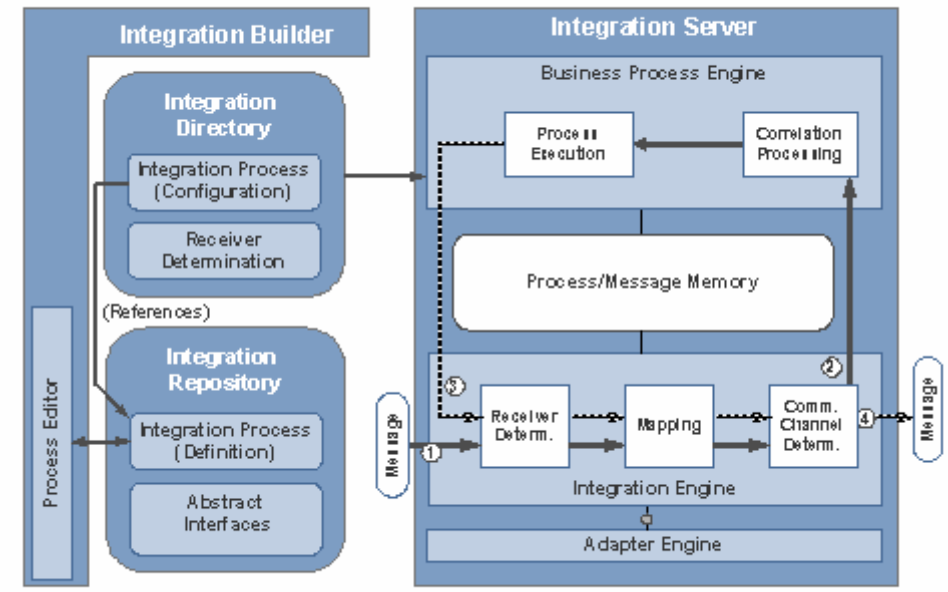

Figure 5. SAP XI Architecture

SAP XI – now being renamed to SAP Process Integration (PI) – runs on the SAP Web Application Server (SAP Web AS) component. SAP XI reduces integration and maintenance costs of IT systems by providing a common, central repository for interfaces. It supports cross-component business process management (BPM) within the same solution. And, it offers an integrated tool set to help organizations build their own integration scenarios by defining the appropriate messaging interfaces, mappings, and routing rules.

2.2.3 Solution Manager

The SAP Solution Manager is a platform which provides integrated support of the life-cycle of a business solution, from the Business Blueprint via configuration to productive operation. It provides central access to tools, methods and preconfigured business contents, which can be used during evaluation and implementation, as well as in operation processing of SAP systems. In addition, a user may create his own project templates, which can be reused in an implementation via an authoring function.

The Solution Manager is technically an add-on for the SAP Web Application Server since release 6.2. It supports a user in all stages of evaluation, implementation and operation of a SAP system. In detail, these stages include tasks like project preparation, evaluation and description of business scenarios and processes, the configuration, comparison and distribution of project-specific customizations, the setup and execution of scenario tests as well as project analyses.

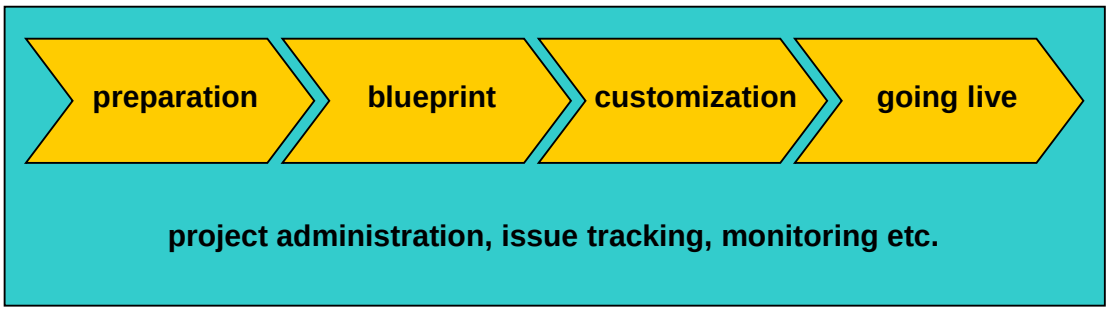

Figure 6: SAP Solution Manager stages

There are four main stages in the process of developing a solution via the Solution Manager.

1. The first is called *project preparation*, which includes the definition of the actual project and the setup of the system landscape.
2. The second is the *business blueprint*. In this stage, a solution based on SAP processes will be defined by analysing the customers requirements.
3. It is followed by *configuration/customization*. All processes defined in the earlier stage will be configured and all customization settings will be synchronized among all systems in the system landscape.
4. The fourth and last stage is the *final preparation and going live stage*. Here, all defined processes will be tested and training will be performed.

Above all stages, project management, issue tracking, monitoring and reporting as well as the definition of roadmaps for subsequent steps are applied/utilized. The most interesting and most important in the context of variability management is obviously the third stage: customization.

The Solution Manager gives access to several tools. Important in this context are the following: the Implementation Guide, Business Configuration Sets and the Customizing Scout. Solutions created with the Solution Manager serve as templates for the actual implementation of a customer solution. Several templates for common business cases are already provided by SAP and are delivered with each installation of an SAP system. In addition these templates can also be created by customers and (re)used in several implementation projects. They can also be transported and distributed among systems for use as a basis for actual projects (for instance in a global roll-out). SAP partners may also create templates in customer projects.

Business Configuration Sets are predefined system configurations and include all settings for a solution. They can be created, ex- and imported via the Solution Manager.

The *Customizing Scout* is used for system migration. It compares customized objects inside a system in a mySAP system landscape with an R/3 system. This is done by examining a reference (source) system and several target systems. The Customizing Scout allows the distribution of customization settings between the source and the target systems. Several different modes of synchronization are possible: initial download, timed, automatically after change, and manual.

2.2.4 Implementation Guide for R/3 Customizing (IMG)

The Implementation Guide (IMG) allows the customization of selected business processes. It lists all necessary and optional actions required for implementing an SAP system. Its primary purpose is to allow a user to control and document the whole implementation process. It is also used for making customer-specific settings in an SAP system.

The base is the *Reference IMG*, which contains all IMG activities and relevant documentation. It covers all topics of an SAP system, for example, enterprise structure, financial accounting, controlling, materials management or production planning. The IMG guides the attention of a user on which configuration options exist and which need to be used for certain application fields.

The Implementation Guide is structured hierarchically, its structure follows the hierarchy of the application components (i.e. *Recruitment* is located under *Personnel Management*). The central parts are so-called IMG activities that enable ways to customization and perform important system configuration tasks. The implementation team accesses the documentation part of the IMG to perform settings in an actual project via the IMG.

2.2.5 Enhancement Framework

The Enhancement Framework (EF) was designed to overcome older techniques to enable users to modify the standard behaviour of an SAP system. The EF tries to combine the easy maintainability of standard software with the high flexibility of proprietary solutions while avoiding the drawbacks of both (lack of flexibility in standard and upgrade issues in customized software). The EF is not a single mechanism; instead it is the integration of various techniques for modifying development objects.

In previous releases of the SAP system, there were predefined points at which users were able to insert so-called *modifications*. This procedure was supported by a *Modification Assistant*, which was able to observe user add-ons (up to a certain degree). There are several shortcomings that are connected to these modifications:

1. There is no support for system upgrades; an upgrade may render modifications unusable.
2. It is quite difficult to trace developments made in different parallel system back to one central system.
3. There is a high cost for testing systems with a lot of user modifications.

The Enhancement Framework has been introduced in SAP NetWeaver 2004s, Release 7.0, and aims to unify possible types of modifications/enhancements as well as organize enhancements as effectively as possible. At the core of the framework there is a simple structure consisting of a hook and an element that can be attached to this hook. The EF is supported by a dedicated tool, the *Enhancement Builder*.

The main function of the EF is the modification, replacement and enhancement of repository objects and foreign objects – objects that form the technical basis of an SAP system. Control over these objects is provided via the *Switch Framework*, which is explained in more detail in another section below.

There are three elementary concepts in the Enhancement Framework for modifying/enhancing development objects:

1. *Enhancement Options* (EO) defined as positions in repository objects, where enhancements can be made. Two types of EO exist: explicit options and implicit. An explicit option is created when points or sections in source code of ABAP programs are explicitly flagged as extensible. These options are *managed* by Enhancement Spots and *filled* by Enhancement Implementations.

In contrast to explicit options, implicit options are special points in ABAP programs, which can be enhanced. Examples for such special points are the end of a program or the beginning of a method. Implicit options can be enhanced by source code, additional parameters for the interface of function modules or global classes.

2. *Enhancement Spots* (ES) are used to manage explicit Enhancement Options and carry information about the actual position of possible options. A spot can manage more than one option. ES are directly supported by the Enhancement Builder which is integrated in the ABAP Workbench.
3. *Enhancement Implementations* (EI) are the counterpart for ES. At runtime one or more EI can be assigned to a single ES. There are several types EI: Source Code Enhancements, Function Module Enhancements and Global Class Enhancements. Source Code Enhancements represent the direct insertion of source code at predefined locations in ABAP programs. These locations can be defined by implicit and explicit Enhancement Options. Function Module Enhancements represent the enhancement of parameter interfaces. For example a new optional parameter can be added to the interface of a function module. In addition via Global Class Enhancements new attributes can be added to repository objects or special pre-/post-methods can be realized, which are called directly before/after ABAP methods.

Obviously, these concepts can be roughly compared to concepts of **Aspect-Oriented Programming**: Enhancement Options resemble *Pointcuts*, Enhancement Spots map to *Join Points*, and Enhancement Implementations to *Advices*. An example is shown in Figure 7. In this example a simple program is extended by several enhancement implementations. Enhancement 1 is inserted at the position marked with ENHANCEMENT-POINT and can optionally be overwritten by Enhancement 2. In contrast Enhancement 3 is not inserted at some particular point, but replaces a section marked with ENHANCEMENT-SECTION.

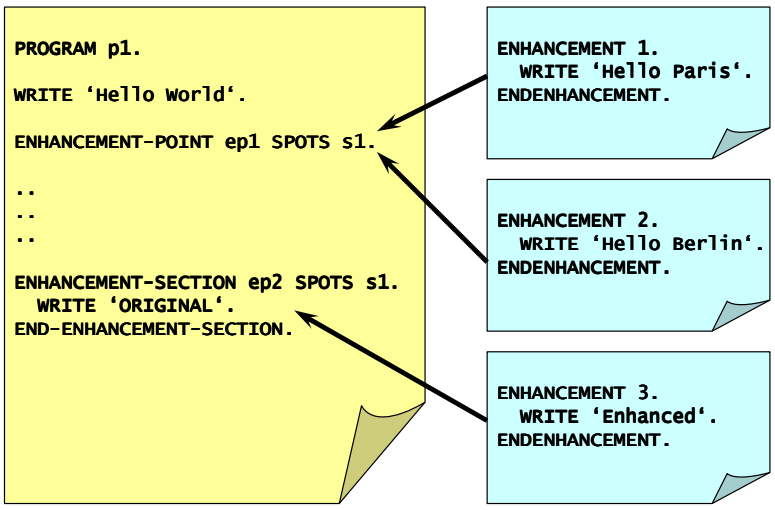

Figure 7 example for an ABAP code enhancement

2.2.6 Business Add-Ins (BAdI)

SAP Business Add-Ins (BAdIs) are one of the most important technologies to adapt SAP software to specific requirements. BAdIs were introduced in Release 4.6 in order to replace function exits. As of Release 7.0 they are part of the enhancement framework. They are realized as explicit *Enhancement Options* (so-called classic

BAdIs). New BAdIs are directly supported by the ABAP runtime environment through dedicated ABAP statements.
BAdIs are the basis for *object plugins* that modularize function enhancements in ABAP programs. There is an explicit distinction between the definition and the actual implementation of BAdIs. The definition of a BAdI contains an interface, a set of selection filters and settings for runtime behaviour. The implementation contains a class implementing the interface and a condition imposed by the filters. An example of a BAdI structure can be seen in Figure 8. In this example a BAdI A may be used for tax calculation. The defintion of this procedure is made in the Enhancement Spot for the BAdI, while the actual calculation logic can be found in Implementation 1 for BAdI A. There may be more than one (two in this example) implementations for the defintion, which can be used mutually exclusive.

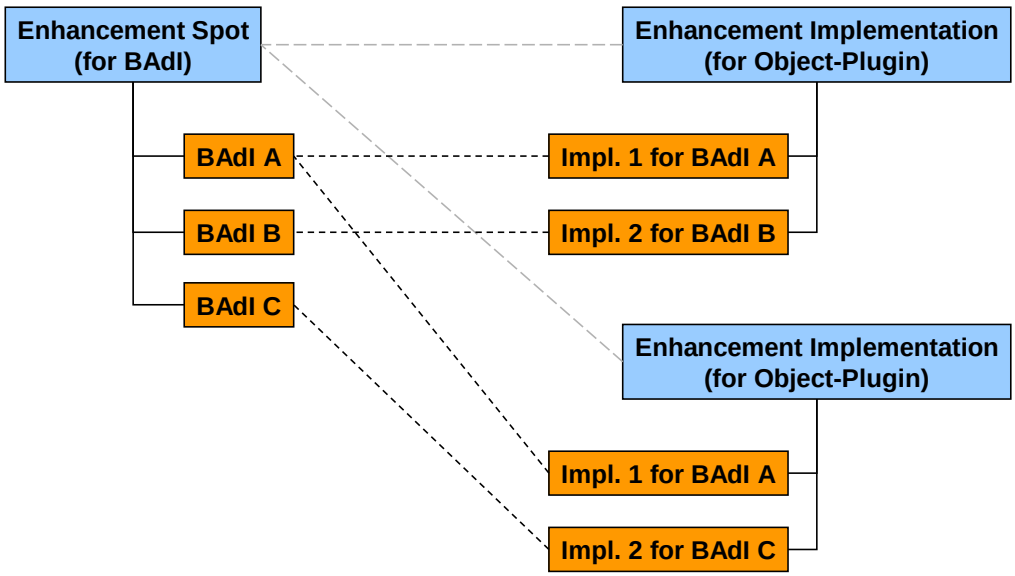

Figure 8 structure for Business Add-ins

Clearly this mechanism is not related to Aspect-Oriented Porgramming, rather it resembles patterns from Object-Oriented Programming, where certain behaviours are defined via interfaces and implemented by a combination of abstract and concrete classes.

2.2.7 Switch Framework

The Switch Framework (SF) allows the control of the visibility of repository objects or their components by means of switches. The SF is integrated in the ABAP workbench and works closely together with the Enhancement Framework. While the Enhancement Framework enables and supports the actual implementation of solutions, the SF controls which of those implementations are finally utilized.
The main purpose of the SF is the simplification of an ABAP-based system landscape by adopting one or more industry solutions in a standard system. Solutions are delivered with all objects/functions deactivated, only appropriate objects are activated on demand. For this reason, the Switch Framework is a modification-free enhancement concept.
The basis of the SF are three main components:

1. A *Business Function Set* (BFS) is a set of Business Functions and corresponds to an industry solution. Inside a SAP system several BFS may exist, but only one may be active at a time.

2. A *Business Function* (BF) is a self-contained function from a business perspective and consists of a set of switches. A BF is some kind of building block for BFS, activating a BF means activating all its switches.
3. A *Switch* is the elementary component in this context; it is a repository object that is able to control the visibility of other repository objects. This applies to single objects like screens or collection of objects like a package. A switch can be assigned to several Business Functions and vice versa several switches can be assigned to one Business Function. A conflict arises if two switches turn on objects that may not be used together. This situation is resolved by special conflict switches and appropriate conflict-resolving enhancement implementations.

The relations between those elements are shown in Figure 9. In this example the BFS contains five BF, where the first and the fourth are activated. Both trigger appropriate switches, which leads to the application of a certain package and some arbitrary component. The whole structure is similar to feature trees, although there is only a limited depth of two or three levels, depending on how fine- or coarse-grained a feature is defined.

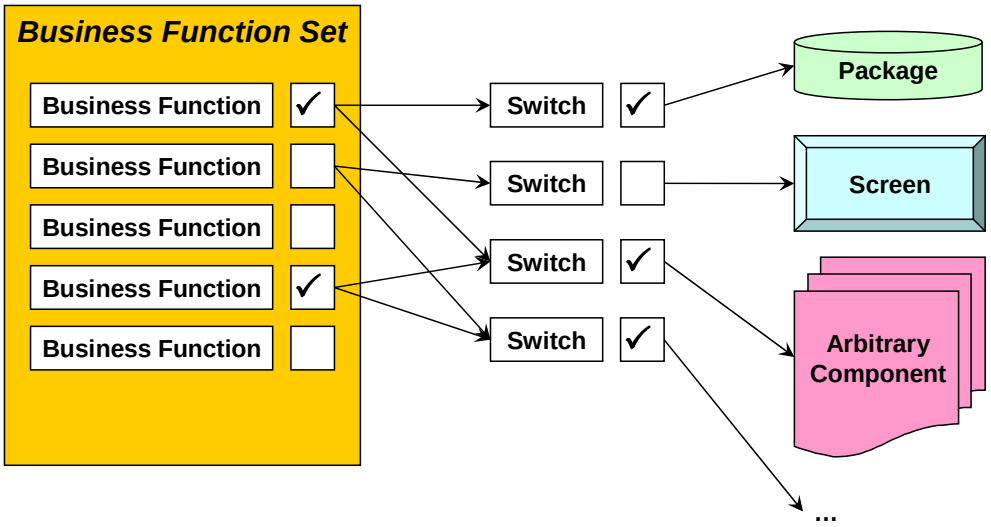

Figure 9 structure of a Business Function Set

The whole configuration of a Business Function Set is stored in so-called *Switch Business Configurations* (SBC). These are data containers with database table entries for industry solutions. Such solutions may contain several SBC, which can be activated in subsequent systems of the solution.

There is a differentiation between *industry* BFS (*industry extensions*) and *generic* BFS (*enterprise extensions*). The Switch Framework can activate exactly one industry BFS, but several generic BFS. Examples for industry extensions are media, telecommunications or oil & gas, as examples for enterprise extensions financial services, global trade or human resources may be mentioned.

Table 1. Comparison of mechanisms in the SAP ABAP stack

	SAP Enhancement Framework (EF)	**SAP Business AddIns (BA)**	**SAP Switch Framework (SF)**
Concept			
Reusable Assets	ABAP Code	ABAP Code	Business Function

Variation Type	ABAP Code Fragments	Business AddIn	Set of Switches
Transformation Type	Refinement	Refinement	Composition
Granularity	Fine, on code level	Fine to Coarse, similar to component level	Coarse, on business logic level
Modularity			
Level of Separation	Structural	Structural	Structural
Dependency on Variation	Unaware	Unaware	Unaware
Binding Model			
Binding Time	Runtime	Runtime	Startup Time
Availability Time	Runtime	Runtime	Startup Time
Scope of Binding	Program	Program	Program
Modularity			
Asset/Variation Dependency	Stable Abstraction	Stable Abstraction	Stable Abstraction
Decomposition of Assets	Possible	Possible	Impossible
Decomposition of Variations	Possible	Impossible	Impossible
Efficiency			
Runtime Overhead	Highly dependent on discovery of enhancements, medium/high	Highly dependent on discovery of enhancements, medium/high	Implementation dependent, low
Memory Overhead	low	low	unknown
Compilation Effort	low	low	low
Other Criteria			
Complexity	low	low	high
Infrastructural Code	low, marking of enhancement points	low, definition of enhancement spots	unknown
Tool Support	yes, via Enhancement Builder	yes, integrated in ABAP Workbench	yes, integrated in ABAP Workbench
Tracing Support	no	no	no

The table above compares three SAP techniques by several important criteria defined in [Poh07].

The concepts of the techniques are different, depending on the level of abstraction they are used to vary existing functionality. While EF and BA allow variations on a code level, SF has got a notion of variation on a higher abstraction level, although this technique is also an implementation technique. While the first two allow refinements, the SF can be used for compositional variations. For this reason the granularity is coarser.

In terms of modularity all three approaches are looking alike. The concerns are structured into separate modules without clear relations between each other. In

addition the reusable code is unaware of possible variations and will work without taking the functionality of potential extensions into account.

All approaches support the concept of late binding, that is, resolution takes place at startup- resp. runtime. For EF and BA variability is resolved at runtime, while SF relies on a database containing the values for switches which are evaluated at startup time. In addition the actual variations must be initially present at the same point of time, which allows a decoupled development of assets and variations.

All three approaches feature stable abstractions; unlike in AOP code injections at arbitrary positions are not possible. The decomposability is different in each technique; both assets and variations may or may not be decomposed.

Statements about the efficiency of the approaches are relative. Usually the runtime overhead in dynamic techniques like EF and BA are higher than in static approaches. The runtime overhead for SW is dependent from the actual implementation and also depends on the underlying database containing the value of the switches. The same is valid for statements about the memory overhead. Compilation effort is in every case low.

All approaches are supported by dedicated tools, but lack tracing support. The complexity is connected directly with the abstraction layer of the variations.

2.2.8 Business Rule Engines

A key property of SAP customers is that every business is different. Although there are many common parts (predefined business content and built in business best practice are actually major reasons why customers buy SAP software), most companies draw their competitive advantages out of subtle deviations from standard business processes. These variations often go beyond simply enabling/disabling switches or changing parameter values. Business experts need means for “programming in the large”, i.e., wiring state transitions and message-based process interactions, and “programming in the small”, i.e., being able to model conditional and/or parallel execution of business process steps, ideally supported by graphical tools.

The Business Process Execution Language (BPEL) [BPEL07] was standardized by the OASIS group for exactly that purpose. It interacts with external Web Services to orchestrate higher-level business processes out of these building blocks. Graphical tool support for constructing orchestrations is available, for instance, using the Business Process Modeling Notation (BPMN), as a graphical front-end to capture BPEL process descriptions. Numerous BPEL engines from different vendors already exist today for executing BPEL-based process descriptions.

An example for such business rule engines is the Business Process Engine (BPE) as part of SAP XI (see above): The business process engine (BPE) is tightly connected with the integration engine and fully integrated into the integration server. During message flow between heterogeneous systems, the engine makes use of all the shared collaboration knowledge needed to execute a business process. An easy-to-use graphical modeller gives you access to the message types and interfaces involved in a process. It lets you define the series and sequence of steps and actions required to run the process. During execution, the BPE also correlates and links related messages based on a unique, user-defined identifier.

In summary, there is a clear need for flexible configuration/variation of runtime behaviour by business domain experts (i.e., end users without sophisticated

programming skills). Hence, DSLs or other formats for representing executable models are required, which can be dynamically loaded, interpreted and/or compiled at runtime.

2.2.9 SAP Modelling Infrastructure (MOIN)

As an effort to consolidate various different modelling technologies and metamodel repositories, SAP NetWeaver launched project MOIN (Modelling Infrastructure) to implement the platform for SAP's next generation of modelling tools [AHK06].

The MOIN is an implementation of a MOF-compliant repository based on MOF 1.4. It provides persistent storage and query capabilities for models, versioning based on software logistics systems such as DTR or perforce, Java-based access using type-safe and / or generic JMI interfaces, and an XMI import / export. Furthermore, several services such as model transformation and code generation will be offered by MOIN.

Frameworks for the construction of graphical editors or management of graphical notations and corre-sponding diagrams are built on top of the MOIN. At the heart of MOIN is an in-memory cache for model elements that provides various service interfaces to MOIN clients and uses pluggable services to load and store models as well as to manage models.

MOIN will be available for various runtime-environments. Immediate candidates are the SAP J2EE engine and the Eclipse-based IDE SAP NetWeaver Developer Studio. Immediate applications for this technology will be the modelling of business processes for business rule engines as introduced above.

2.2.10 ESOA and next generation Application Platform (AP) modelling

The Application Platform (AP) is the basis for SAP's next generation Business Process Platform [Hei07], which allows developers and solution managers to flexibly build solutions for small, medium, and large enterprises on top of it. Figure 10 depicts, how AP is embedded into the landscape of SAPs business cases. AP itself is the core for implementations of business functionality. It is structured into one foundation for generic functionality and multiple so-called deployment units for business scenario-specific functionality. Deployment units group several related process components that typically run together on one machine of a business partner. Selected deployment units may be replaced by custom-developed deployment units or other applications.

A single deployment unit consists of several process components, which are reusable building blocks for modelling business processes. Communication between deployment units is handled via asynchronous web service invocations. Inside deployment units messages are passed between business objects. A business object is an entity of significance to a business, it encapsulates business data and logic, acts as a service provider and/or consumer. Business objects are described by a business object model which defines the structure, type, aspects of behaviour and service instances of the appropriate business object. The model is used to create a programming language-specific representation. Access to business objects is granted via its service interfaces, which are defined with WSDL. Important questions currently under discussion in this context include the challenge of providing more control and government over industry and customer extensions than with previous extension mechanisms (like the above mentioned Switch Framework).

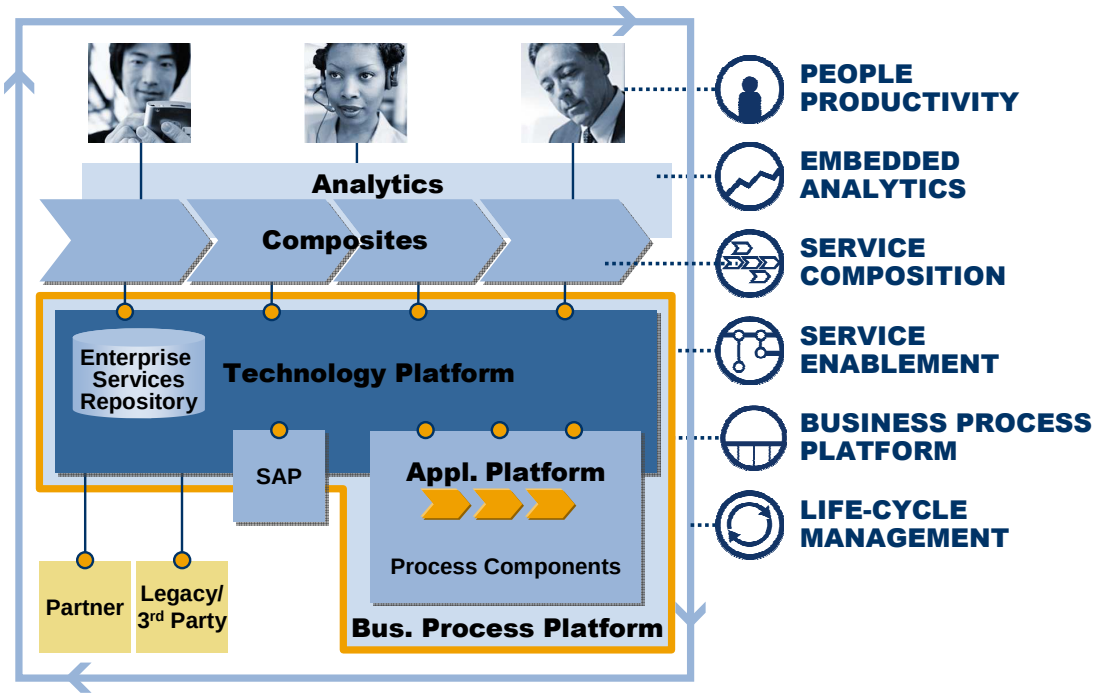

Figure 10 Overview over SAPs Business Process Platform

2.2.11 Product Development Processes and Technologies

The growing diversity of SAP's product portfolio has led to an increased complexity of software configuration management and software product line evolution. As a result development processes and utilized tools have to keep the pace with this kind of progress.

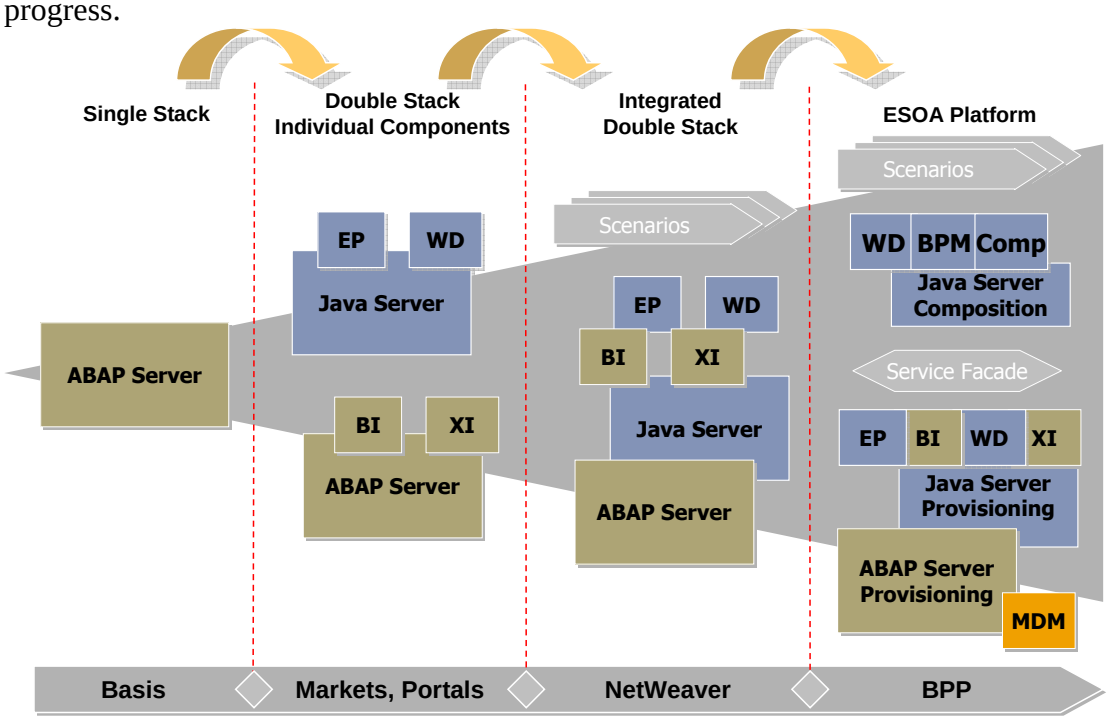

Figure 11 Evolution of the Software Stack at SAP

Figure 11 illustrates the evolution of the software stack over several years. Starting with a single ABAP-based stack the software basis has grown into a fully fledged platform for Enterprise Services. The increased complexity is obvious and has impacts on Stack Management (commonalities, variations, and dependencies),

Software Lifecycle Management (traceability throughout consecutive development stages) and Version Management (compatibility and integration of component versions in platform stacks).
The development of features in the SAP system landscape follows the Product Innovation Lifecycle (PIL). PIL subdivides the whole product lifecycle into several stages, reaching from the initial idea to deployment and maintenance. It is an interactive process, unlike other development processes that lean back against the classical waterfall model, PIL contains iteration cycles that may be passed several times and is supported by several tools, which as such are only partially available as commercial products.
PIL incorporates several software configuration management actions. However, many of these actions were created out of best practice and without standardisation. Furthermore development documents like specification and design documents or test plans are handled outside of formal configuration management.

Software Configuration Management for artefacts developed for the software stack mentioned above is embedded in the PIL activities and may be categorized into three cases:

1. For ABAP programs the SAP Transport Management System (TMS) is used. Software configurations are stored as dedicated SAP systems.
2. Most non-ABAP configurations are managed with Perforce. Software configurations are typically accessible on so-called branches using a time-stamp or related information to denote the time-dependent state of the configuration.
3. Some Java projects use the SAP Design Time Repository (DTR) which is integrated into the central SAP development landscape supporting the entire software lifecycle. The DTR is a system for version control, is used to store java code centrally and allow distributed java developments. All source files that are created or changed during the development are stored in the DTR. Changing files necessitates developers to check out relevant files, while they have to be checked in again after the change process.

SAP is currently evaluating new products, methods, and processes for software configuration management in order to speed up development and production and allow for extended use of agile development methodologies like Scrum and Extreme Programming.

2.2.12 Software Evolution

Within the context of software evolution, the issue of data migration is ruled by the Legacy System Migration Workbench (LSMW). LSMW is a tool that supports data migration from legacy systems (non-SAP systems) to SAP systems. Utilizing custom programming for the transformation of data and updating SAP systems is extremely costly, time-consuming and error-prone process.

The LSMW tool makes it possible to transfer data from a variety of sources without any programming, thus counteracting the before mentioned disadvantages. A changeover is basically governed by a set of rules, which has to be defined in advance. The LSMW then uses this definition to generate a program, giving considerable support during the migration. When data is imported, the system performs the same checks as it does during online entry.

The most important benefits of the LSMW include its independency from SAP releases, platforms, and the kind of data to be migrated; tracing of the migration history at any time and maximum quality and consistency through import checks.

During an upgrade or the import of a support package, existing objects of the SAP standard are overwritten with the objects redelivered. To help the customer retain the objects modified in a previous release, SAP provides all modified objects which are redelivered (in an upgrade or Support Package) in the upgrade adjustment. The basic transaction process is supported by specific tools, as follows.

Those object, that have been modified by the customer and those redelivered by SAP during the upgrade or with the support package are presented for adjustment. During the redelivery import, the system automatically recognizes, whether a modification adjustment is required and visualizes its locus. At each modification request, it is necessary to decide, whether to retain ones original adjustments, or to reset them to the original. Otherwise, the new original stays active in the system.
The Modification Assistant supports this process of adopting customer modifications.

In general, objects altered using the Modification Assistant can now be automatically accepted into the upgraded system, if the modifications undertaken in the original version do not directly overlap those made in the customer version. If collisions occur between the two versions at upgrade (naming collisions, or if SAP has deleted an object modified by a customer), the system offers support during the upgrade, which means that a semi-automatic adjustment is made.

In some cases, however, it is still essential to manually adjust objects. Objects modified according to the old system used prior to the advent of the Modification Assistant must be manually maintained after an upgrade has been run.

2.3 Siemens

Siemens AG is a collection of business units that operate in different domains with different product innovation cycles and different business models. Therefore there is neither one common development process for all Siemens business units, nor one consistent set of development practices for product line engineering. Main differences are

- Solution versus product driven businesses, e.g. postal automation system solutions build on a common set of base assets, but have to be customized heavily for each customer, while telephone switches are standardized products.
- Product/innovation cycles range from a couple of months for e.g. mobile phones up to decades for rail traffic control technology.
- Security and reliability requirements, e.g. medical devices or traffic control systems have to fulfil high reliability and security requirements, while those requirements are a lot less critical for car entertainment systems.

We will give a rough overview over the practices employed in Siemens.

2.3.1 Requirements Engineering

For product driven business new products or product generations (group of products that cover several market segments, like a low, mid and high end computer tomography) are usually developed according to the innovation cycle in the domain (probably a new generation every year). New requirements are collected from different stakeholders, e.g. telecom providers send requirements list to all suppliers or product managers visit customers. These requirements are condensed into features; the relationship between requirements and features is maintained in tables or in requirements management tools like Doors or RequisitePro. Tables list the configurations for the instances of the family, i.e. which features will be available in which products. Sometimes this information is also cast into a feature tree.
Product driven business allows a pro-active product line approach, where requirements are implemented in base assets and products are derived from these base assets.

For solution driven businesses requirements usually come from a specific customer for every instance of the product family. It is in the interest of the solution provider to reuse as much as possible of pre-existing base assets for a new customer, because this makes instantiation cheaper and the provider more competitive. Therefore it is necessary to have a good understanding of the features already provided in base assets. This information is kept in feature lists and feature trees. Requirements that are not already implemented in the base assets are usually implemented for the customer specific solution and requirements that result in features, which can potentially be reused for other customers are then added to the base assets in a re-active manner. The feature information has to be updated accordingly.

2.3.2 Domain Design and Realization

Domain Design is concerned with producing a product line architecture and reusable components that support the variability needed for all products of the product line. This variability support can also be an MDD infrastructure or other means to instantiate products in application engineering.

2.3.2.1 Implementation Techniques for Variability

For efficiently handling a family of software systems in a domain it is essential to know the domain abstractions and to generalize and separate them with stable interfaces. Stable interfaces are the most profound mechanism for reuse and exchangeability of implementations, which is a way to support variability.

Beyond that the following main technical options exist to cope with variations of base assets during software architecture, design and development:

- Another level of indirection—In this category fall the typical design patterns used for decoupling and configuration, such as Factory, Strategy, Extension Interface, Bridge and Adapter, but also general framework principles such as inversion of control and dependency injection, as intensively used by the Spring framework [Fow04]. To avoid the mingling of variations and allow for easy re-configuration, configuration options are externalized into configuration files, where variations can be expressed declaratively. Certain architectural patterns, sometimes also referred to as architectural styles, such as event-based communication and Pipes and Filters architectures allow for more easy variation, as they inherently decouple a system into exchangeable parts.
- Language support—This includes approaches, such as aspect-oriented programming, where variations are encapsulated as aspects, template meta programming, where commonalities are expressed in templates, or domain-specific languages (DSL) combined with code generation. Further, macro languages, such as the C++ #ifdef construct, allow to for compile-time binding in source code.

All of those options are used in Siemens product lines, though generative approaches including AO are still rare.

The typical means are OO in combination with stable interfaces for important varying domain abstractions. A simple example for the latter is hardware abstractions in automation and control systems. Devices like motors, sensors or higher level entities like cameras or conveyor belts, which themselves group sensors and actuators, are represented as abstract interfaces to the machine control software. The gap between the interface provided by the hardware element and the interface required by the software has to be implemented for each device, but typically there are no adaptations to the control software required if new devices of a known type are integrated.

Nevertheless, developers of component-oriented business applications make increasing use of aspect-oriented programming, also within Siemens. Frameworks such as Spring or J2EE compliant containers like JBoss already offer aspect-oriented extensions. The very existence of frameworks, like EJB, and specific design patterns to decouple responsibilities confirms the need for AOP. They were developed to untangle concerns to be able to evolve and reuse infrastructure and business code separately. The advantage of AOP is that it is not limited to a single domain in the way that EJB is limited to server-side component computing [Lad03]. Examples for AO in product lines in Siemens are Spring aspects for security and life cycle management in a platform for telecommunication applications and JBoss AOP in an

IP based communication service for voice, video, unified messaging and instant messaging for service aspects.

The typical usage scenario for generative approaches including MDD are currently either generating glue code for embedding business components in a given platform or for easily formalize-able code like communication code in embedded systems. An example for the former is a DSL and a generator for generating interception proxies for a telecommunication application platform. The DSL allows attaching interceptors to business components, simulating a simple AO infrastructure. Another example is the generation of MOST-bus specific communication code for small controllers in a medical imaging system. Communication partners (device controllers) and the payload are specified in tables, supported by dedicated editors.

2.3.2.2 Binding Variability

Depending on requirements like footprint, security, and runtime flexibility different measures are taken for implementing and binding variability. E.g. telecommunication enterprise applications need runtime configurability and therefore implement component containers and composition filters for flexibly changing the runtime configuration of a system. Automation and drives software requires often small runtime resource footprint, therefore the variability will be bound at load time through configuration files. For high security domains, e.g. train traffic control and supervision systems the code has to be certified by national certification bodies. Variation is only allowed before compile time, so variability usually gets incorporated via #ifdefs. The actual code for a variant is specified by preprocessor defines and must not change after certification. Code for new variants is introduced in new conditional compilation blocks only.

2.3.2.3 Platforms

Business units that do not have a dedicated product line engineering approach nevertheless usually have at least a common base asset for domain specific infrastructure services called a platform. Such platforms typically care for communication, persistence, user interface support, some introspection support like tracing and debugging features and usually typical domain specific extensions like image processing for optical systems.
A common platform is often the first step towards product line engineering, since practices like commonality/variability analysis have to be introduced once the capabilities of a platform reach beyond general purpose middleware responsibilities.

Siemens has several examples of platforms that are the basis for further platforms, e.g. in medical engineering one platform for all imaging systems is the basis for further platforms in product lines for e.g. magnetic resonance systems or computer homographs.

2.3.3 Application Engineering and Product Derivation

For product driven business application engineering and product derivation can be as simple as assembling the product from the pre-built base assets. Often however, and definitely for solution driven business the product/solution hast to be customized or even product/solution specific extensions have to be implemented.

The goal however is to avoid implementation and derive new products mostly through customization. For example in automation systems it is common to have a staged approach for customization. On the top level the layout of an automation system is configured according to the hardware and mechanical capability of a machine. On the next level of configuration machines offer specific functions for calibration, where the machine either automatically or guided by an operator determines reference positions or settings and keeps acquired data for production runs. On the last level, customer specific customizations can be set by "programming" the machine through teach-in or with dedicated domain specific programming languages.

Next to configuration files configuration and build management tools are the state-of-the-art tooling for product derivation. Configuration management tools are used for keeping and managing variations of base assets and allow to assign a label to a set of base assets, and for each of them exactly one version, that then form a base line or a product. Build systems can either use this information or get the information in their own scripting language on where base assets can be found and set pre-compiler variables and compiler switches for generating products.

While those mechanisms are proven technologies, the mapping between the information kept in build scripts and configuration management labels and the information on the feature set selected by those mechanisms is not well supported by tools. This information has to be kept separately.

2.3.4 Traceability

There are several approaches employed for tracing requirements horizontally and vertically in Siemens. For mature domains often requirements management tools like DOORS or RequisitePro are used for capturing traceability information. A light weight way to store tracing information is to include requirements (often only requirements IDs, the requirements themselves are stored in an RM Tool or sometimes also only in Excel tables) in architecture and design documents and test plans. Using a variant management tool like pure::variants for traceability is currently tested in some business units. Business units that have very stringent process requirements, like medical engineering, have their own tracing tools, e.g. MedTrace. The vertical traceability information between base assets (domain engineering level) and products (application engineering level) can usually only be derived from the information available on which base assets are used in a product. This information is usually not documented but contained in a configuration management system and the build system.

2.3.5 Process

Next to the term product line engineering process Siemens business units also use the term platform process, if products share only a common platform and are independent otherwise.

Both kinds of processes are iterative (the development processes for one iteration of a product line or platform are usually also iterative). Iterations are either a new product generation in product driven businesses or a new customer in solution driven businesses. In the former case, product line evolution is often pro-active; the base assets are evolved in domain engineering and then used in application engineering. In

solution businesses evolution is often re-active. First the new solution is built, and then the changes are fed back to the set of base assets.

Depending on the reliability and security requirements, the processes are very well defined and strictly obeyed or not. E.g. in traffic control systems or medical engineering the development process is required to be very strict with complete traceability between all artefacts. Such processes are a pre-requisite to be accepted by national bodies like the Food and Drug Association in the US, which decides over the admission of health care goods in the US.

2.3.6 Product Line Engineering Example of Siemens AG, VDO

Siemens VDO runs a product line for gasoline systems. The domain has very stringent functional and non-functional requirements. The fuel consumption and emission values have to drop constantly while at the same time the performance of the vehicles is expected to rise. This results in increasingly complex systems that have to become cheaper steadily due to market pressure.
Siemens VDO was able to survive in this market by developing and running a strict product line process. The main measures taken to make this PLE process work are

- Architecture
  the architecture is built up in layers, with strict interface definitions between the layers. The bottom layers are hardware, hardware dependent software and an independent hardware abstraction layer. On top of this, functional units, called aggregates, are deployed; they represent the implementation of features in the product line. These aggregates themselves are split into three parts: fixed generic code that is reused in every product without change, a configurable part that represents the deliberately built in variability and a specific part that is coded anew for every product. The functionality in each of these three parts does not remain constant over the lifetime of an aggregate. Specific parts can become configurable or even fixed code parts and vice versa.
- Process
  the development process follows roughly the V-Model, only that this V-Model is replicated and iteratively revised many times in parallel. Every reusable code asset has its own sub-process as well as every product. The crucial part is the synchronization between the domain and the application engineering processes. According to the responsibilities this only works, if there is not only a consumer/producer relationship from the applications to the domain, but also an equally important feedback relationship form the applications to the domain. This relationship has to be deliberately kept up and permanently enforced. Another important point is a properly working information management. Not only the aggregates themselves are kept as base assets, but all the information that led to or is necessary to use the aggregate, e.g. tuning guides, validation reports and simulation models.
- Organization
  to run a product line engineering approach, an organization has to change its culture and this culture change has to be initiated by the management. Domain developers and application developers need to get used to their new roles, especially application developers have to accept their role as system integrators. Like for every PLE approach, also VDO faces the challenge how to keep domain and application engineering synchronized. The measure is to

have a platform team that is responsible for the architecture for both, the domain and the applications, and runs the process of coordinating both sub-processes.

2.3.7 Current Industrial Practices in Siemens Concerning D1.1, D2.1, M3.1, and M4.1

The following listings of practices and tools are not complete. Siemens has a number of business units and these business units are again divided into smaller units that develop product families. There is a huge diversity between all these groups and there are doubtless practices and tools employed we do not know of.

2.3.7.1 Current Industrial Practices in Requirements Engineering regarding D1.1

The following practices can be observed for requirements engineering:

- Siemens business units use FODA as a means to describe the variations in their product portfolios. Sometimes there is no feature modelling notation used because tool support is lacking, but the information is presented textually in tables. However, there are some first attempts to cast the information into pure::variants.
- Use cases are very common in Siemens to give a higher level view on requirements. Since standard UML use case diagrams do not support expressing variation, the variation is typically expressed in several instances of a diagram or if there is a textual description of the use case, variations are an extra sub section.
- Viewpoint based approaches are rarely used, but they are used, e.g. in Medical Engineering. Stakeholders are typically stakeholders in a hospital, like doctors, patients, support personnel on the one hand and the hardware and software development stakeholders on the other.

Tools that are used for Requirements Engineering are

- Doors: increasingly used, nearly becoming the standard tool
- RequisitePro: was the de-facto standard once, but decreasing
- Excel Sheets: still used in many places
- Documents (MSWord): also still used when the number of requirements is not high or in combination with Excel sheets.

Some business units start to use pure::variants for expressing the variation in their product portfolio.

2.3.7.2 Current Industrial Practices in Architecture regarding D2.1

For documenting architecture, also a product line reference architecture, typically standard UML is used within Siemens, usually in combination with textual documentation. Entities in such an architecture are sometimes described using CRC cards. UML extensions that help expressing variation are not used, since such extensions are typically not tool supported. We are not aware of any usage of Architecture Description Languages.

Domain specific languages for simply documenting an architecture are not common within Siemens. At the places where MDD is used, the domain specific languages are

either textual or UML with stereotypes. Currently some business units investigate Microsoft's SoftwareFactory approach, which is Microsoft's MDD support. Typically these models are directly transformed to code or to configuration information (configuration files, deployment descriptors, build files ..).

Architecture Reviews are quite common in Siemens, especially for product lines. They follow scenario based approaches like ATAM or experience-based reviewing techniques.

2.3.7.3 Current Industrial Practices in Implementation regarding M3.1

Some Siemens business units use pure::variants. Gears is not used yet, but there is interest in the tool.

2.3.7.4 Current Industrial Practices in Traceability regarding M4.1

Concerning traceability, chapter 5.2.2 of M4.1 lists the traceability approaches used in Siemens, which are

- Requirements engineering tools like Doors
- Traceability information in documents
- Variant management tool pure::variants
- Asset repository
- Dedicated traceability tools like MedTrace

3. Analysis and comparison of approaches

In contrast to the previous chapter, we will analyse here what the discovered practices have in common for the purpose of identifying synergies for applying AMPLE concepts of work packages 1-4.

3.1 Commonalities and differentiators

When examining the practices employed at the industrial partners of AMPLE, it is obvious that the technologies for handling commonalities and differentiators depend largely on the appropriate application domain. The landscape of applied technologies is heterogeneous. It seems to be quite difficult to identify techniques that are applied in favour of others.

SAP bases its product family on two different language stacks (ABAP and Java). While Java is widely used throughout various application domains, ABAP is tightly bound to SAPs business model and its associated products. For implementing business logic and supporting functionality (i.e. communication, persistence or UI) several publicly available frameworks are used when working on the Java stack, proprietary techniques have been developed for the ABAP side. Many of the current problems originate from this heterogeneity. Integrating both language stacks closer at virtual machine (VM) level seems promising. This is however out of scope for AMPLE. This question is rather, how to organise dependencies between core assets of both stacks more appropriately.

In the context of supporting variability there are several technologies noteworthy. Business AddIns can be understood as some kind of plug-in mechanism. It mimics the common properties of framework technologies like extending the functionality of a

host application, registration at the host application to enable activation and defining some kind of protocol for communication purposes.

The runtime configuration of SAP applications is usually done based on entries in database tables, allowing the late activation of application modules (see the section describing the Switch Framework above). This approach can clearly assigned to the group of techniques where variability is bound at the late point in time.

Both technologies provide diametrically opposed functionality. While configuration of additional features via Switch Framework provides positive variability handling, database-table-based configuration turns off unnecessary features (negative variability handling). Both are late bound; the opposite, the generative way (early binding of variability) is rather uncommon at SAP.

For handling the whole process of developing software systems SAP defined its own process: PIL. The process incorporates all stages of software development, therefore any activities related to software product lines have to be integrated into this process. The stages of PIL are supported by dedicated tools, some of them are available commercially. Variability handling is not explicitly incorporated in the PIL process, for this reason the appropriate tools do not explicitly supported variations. In addition PIL is not a model-driven process; it only employs model-based concepts. This leaves room for concepts to be developed in AMPLE.

For software configuration management mainly Perforce is used in production environments. In addition some other systems are applied internally (i.e. Subversion). In terms of SCM (Software Configuration Management), there is no difference between industrial partners in AMPLE.

As indicated before model-driven concepts aren't common in SAP. State of the art is rather model-based, than model-driven design. UML is used throughout the companies processes, but the use is often narrowed to documentation purposes. In some projects UML is also used to actively drive the development process, but without additional support like UML profiles or extensions for variability/commonality modelling. These aspects have to captured by hand in (mostly) textual form.

Dedicated Architecture Description Languages aren't used, instead architectures are described via Fundamental Modeling Concepts (FMC). This notation is used for high-level architetural block diagrams and contains support for modelling structural variance. FMC is very common, but, however, it is mainly used (just like UML) for documentation purposes. A tool set that is driven in some way by FMC diagrams does not exist. From SAPs viewpoint, it can be confirmed that ADLs (Architecture Description Languages) failed to gain industrial momentum. Instead, SAP interprets ADLs as domain specific languages. In this context SAP follows a model-driven approach, where metamodels and DSLs are defined and transformations on that basis are developed. Model-based weaving is probably the most promising way to integrate techniques originating from AOSD into this landscape.

Another important aspect when examining the applied technologies of the industrial partners of AMPLE is tracing. The problem of traceability is not any longer only of academic interest - it has also been perceived in industry in the meantime. According

to an internal audit of customers of SAP, missing traceability during the whole development cycle is the top-rated weakness. In this audit, which has been conducted in 2004, 10 out of 11 customers criticized this issue. In addition, missing traceability information has been explicitly mentioned as weakness in 2005 in an external ISO certification audit.

An end-to-end strategy for tracing artefacts created during the development cycle of products (and product lines) is currently missing for SAP. SAPs internal process PIL supports the tracing of artefacts partially only. Market and software requirements can be set into relationship and can be bound to test cases. In contrast, explicit support for models or model elements is missing, because PIL is not tailored to model-driven design.

While at SAP the landscape of applied techniques and supporting tools is in some sense quite homogeneous, Siemens may be consulted as an opposing example. Inside Siemens there is no consistent development process or specific practices for product line engineering. Due to the heterogeneity of products and solutions produced in Siemens, one consistent development process or a handful of common practices for PLE are not realistic. The Siemens business units work in different domains with different product life cycles, market requirements maturity. Overall, a broad range of business requirements has to be supported, i.e. solution vs. product driven development, where only a single product instance exists, tailored to specific customer needs while on the other hands products are designed for a mass market. Both, however, base on certain core assets, but these may be designed in a completely different way. This is also reflected in other boundary conditions like short vs. long product lifecycle, high vs. low reliability/security requirements. However, product line engineering as a concept is interesting for nearly all business units.

Of course, there are some techniques that applied more frequently. Design patterns are used for decoupling. Configuration is often done via configuration files, which shows some commonality to SAPs approach to use database tables for configuration. It can be pointed out, that techniques for late binding of variability seem to be more popular. Generative approaches including aspect-oriented techniques are rarely used. On the other hand the point of time for binding variabilty often depends on the concrete product category. Like SAP, many business units in Siemens uses component technologies together with textual configuration information that is read at deployment or even at runtime to achieve flexibility. However, for domains where runtime resources are restricted, variation points need to be bound at compile time. Currently conditional compilation is still state of the art here.

Both worlds could benefit from additional variability support like aspects or generative programming. Especially for resource restricted systems generative programming as used in MDD doesn’t require the variability support in the runtime infrastructure. Variability is bound before compile time.

Configuration for a product in application engineering is typically done with configuration information in textual form. Editing and understanding this configuration information is hard, because the link to the requirements that led to this configuration information is typically missing. Appropriate traceability could help here. Additionally these text representations of configuration information are hard to understand. Constraints between variants can not be checked, inconsistencies often

are not obvious before deployment. Here a combination of MDD and AOSD could help to identify the constraints between variations and to build proper support for variability binding and consistency checks. Along with that, traceability is essential for the evolution of a product line. Reusable base assets will quickly become unusable when the information on how they are linked to other base assets is missing. Integrated tool support for traceability is missing.

Model driven development is increasingly used in Siemens projects, but typically only in restricted sub domains of a product or solution development, e.g. for generating error codes or communication code. AOSD is occasionally used, but typically only for development concerns like traceability or architectural checks. The reason for this is that these technologies are typically not yet well integrated with existing processes and tool chains. In addition it may be observed that the better especially code artefacts are separated and configurable, the harder it is to instantiate products without tool support. Tool support integrating model driven development and AOSD is missing throughout the tool landscape of the industrial partners. AMPLE shows how these technologies can be integrated with product line engineering and with state of the art tooling.

Despite the heterogeneous development landscapes of the industrial partners some commonalities, shared among them, may be identified. Techniques for the modularization of reusable artefacts are very important in the context of software product lines. Such techniques, e.g. using polymorphism in programming languages, component technologies or generative programming, are applied in all participating companies, but impovement is still needed. Contemporary OO techniques and component technologies are often complex and don't offer the degeree of modularization demanded by highly configurable product lines. Generative approaches are beginning to penetrate the industry, but are by far not common and still too demanding for the mainstream.

3.2 Potential for improvement by applying AMPLE concepts

In the future SAP will strengthen its activities in model-driven design, because the conviction that this approach will shorten development cycles and improve the quality of design and implementation has started in gain ground in industrial areas. Model-driven techniques are in fact applied by development teams without being aware of it. A key driver to the enforcement of this trend will be a consolidation of already existing tools in one common platform.

Such a common platform is currently under active development inside SAP. The platform bears the name MOIN (Modelling Infrastructure) and is SAPs effort to implement the base for its next generation of modelling tools. Backed by this modeling infrastructure SAP plans to realize and support large-scale MDSD scenarios, that bring up additional requirements to the development processes employed currently. The implementation of MOIN as the technical basis is only one facet in this context, the development of processes, concepts and tools based not only on this platform, but on model-driven concepts is another. Here, the outcome of the AMPLE project will provide a valuable, input, especially to the traceability of

evolving artefacts in such model-driven development processes and the adoption of aspect-oriented concepts.

An important principle in SAPs business model is the evolutionary development of products. Seamless migration from existing systems to current ones is a key feature for most customers of SAP. The handling of configuration data and product customizations or enhancements have to be subordinates under this premise. At the moment this is achieved by managing several separate code lines per product or even product variation. In addition a large number of possibilities for runtime configuration exist. With a continuously increasing number of customer-specific versions and extensions this approach will become harder and harder to maintain. It is obvious that the scalability of the process is limited. More flexible mechanisms for extending core functionality and configuring this functionality at customer sites is required. The AMPLE approach to defining and developing product lines will provide valuable input to the questions that arise from this scenario.

Overall, it can be noted, that this evolutionary way of maintaining single products will lead to an increase in costs, which conforms to the statement of Lehman et. al., that up to 80% of the lifetime expenditure on a software system may be spent on the activities of change and evolution [Leh01]. However, as Bachmann and Bass observe [Bac01], change may not always be carried out by the original architects of the software system; hence explicit documentation of anticipated variability is required (e.g. where the extension points are located, and how they can be used). For this reason SAP has got a motivation of handing over the maintenance of 3rd party extensions to independent software vendors. Key to this business case are stable concepts for managing products and their customer-specific extensions. Some of these necessary concepts should be the outcome of AMPLE.

From experience in various kinds of businesses, Siemens derives several challenges, for which it is expected, that the AMPLE project will present solution proposals:
The first is mining and documenting variation. Mining variants for a family of products from customer requirements and domain knowledge is a task that is typically not supported, as well as an easy accessible way of documenting this information. This open issue is closely related to the activities related to tracing, undertaken in work package 4 of the AMPLE project. Another area, also closely related to these activities is traceability from requirements to implementation and from domain to application engineering and back: traceability information is essential in the presence of evolution. A product line will only pay off if it survives evolution and traceability is a valuable tool to support this evolution process.
The typical variability mechanisms in design and implementation are manifold and typically hard to manage and to link with the feature or requirements view of the problem space. Here, statements regarding the degree of manageability of variability mechanisms would be helpful.
Typically there is a wide range of variability in a platform, but instantiating a product from this variability is often hard and error prone. Siemens will benefit from tools and conceptual support in this area of application engineering.

The expected outcome for Holos differs from the items mentioned above. It’s important to point out that the structure of Holos is different from the other industrial partner company profiles. Holos is a SME and works with a more flexible methodology that supports the company in achieving good quality results. The

development team dimension and the project cycle timings cannot support a more traditional approach with existence and creation of a great quantity of support artefacts for the project.

Holos' objective is the application of AMPLE results on current and future projects and to support on the introduction of these results in the software development process of the European Space Agency (ESA).

To this end, the case study proposed by Holos is currently on an evolutionary phase, with the birth of SEISOP project, where the Data Processing Model is included. At this moment, the development team and the AMPLE team are working together to define the possible usage of some results achieved by AMPLE on the SEISOP project. With this, Holos expect not only to internally improve the development cycle but also to introduce the methodologies to the Mission Control Technologies Unit of ESA at ESOC who contributes to the investigation of feasible concepts for future missions and the application of new technologies for ESOC core business that is one of the main concerns of ESA team.

4. Conclusions and next steps

This chapter summarises the findings and outlines follow-up actions for Task 6.2 and WP5 with respect to conducting experiments on applying AMPLE approaches to improve the identified processes and practices. The industrial challenges that have been identified can be summarised as follows:

Evolution of product lines and their (potentially aspect-oriented) extensions plays a crucial role for industrial adoption. Maintainability concerns are the primary obstacles. With respect to staged development scenarios (including platform providers, independent software vendors, customers, etc.), such extensions need to evolve seamlessly from release to release. Research is needed to manage dependencies of aspects to base code more explicitly.

Model-driven development has a lot of momentum in industry. One way to gradually introduce aspect-oriented concepts could via this technology. In this context, traceability of artefacts in general (from requirements to implemented feature and from domain to application) and tools for tracing in particular should be leveraged to demonstrate the value of such process extensions. Traceability is also important for mining variants of product families out of existing artefacts.

Last not least, appropriate application engineering tools would help to keep the plethora of variation mechanisms manageable. In this regard, the "impedance mismatch" between rather abstract variability feature models and concrete variation point in architectural models needs to be bridged.

The industrial requirements elaborated in section 3.2 will be taken up in the representative case studies of WP5, which in turn will be used for experiments to understand the impact on existing processes. This will ultimately lead to a generalized software process improvement framework for evolving existing processes to incorporate the AMPLE concepts, which is to be investigated in task 6.3.

References

[AHK06] Michael Altenhofen, Thomas Hettel, Stefan Kusterer: *OCL Support in an industrial environment,* in Proceedings 6th OCL Workshop at the UML/MoDELS Conferences, 2006.

[Bac01] F. Bachmann and L. Bass: *Managing variability in software architectures,* in Proceedings of the 2001 Symposium on Software Reusability (SSR) 126-132, Toronto, (Ontario, Canada), May 2001

[BPEL07] *WS-BPEL 2.0 Specification*, OASIS Standard. 2007, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[Bus03] C. Bussler: *B2B Integration - Concepts and Architecture*, Springer, 2003

[Cla06] E. Clayberg, and D. Rubel: *Eclipse. Building Commercial-Quality Plug-Ins*, Addison Wesley, 2006

[Fow04] M. Fowler: *Inversion of Control Containers and Dependency Injection pattern*, http://www.martinfowler.com/articles/injection.html, 2004

[Hei07] Robert Heidasch: *Get ready for the next generation of SAP business applications based on the Enterprise Service-Oriented Architecture (Enterprise SOA),* https://www.sdn.sap.com/irj/sdn/go/portal/prtroot/docs/library/uuid/00a06116-b134-2a10-0797-9cbb090a903f

[Kel07] Horst Keller and Sascha Krüger: *ABAP Objects. ABAP-Programming in SAP NetWeaver*, Galileo Press. 2007

[Kha07] Safoora Shakil Khan, Awais Rashid, Arda Goknil, Ismenia Galvao, Iris Groher, Christa Schwanninger, Jean-Claude Royer, Kelly Garces, Christoph Pohl: *State-of-the-art for traceability in software product line development, with specific focus on aspect traceability in the software development process*, AMPLE Milestone M4.1. Mar 30, 2007.

[Lad03] Ramnivas Laddad: *AspectJ in Action: Practical Aspect-Oriented Programming*, Manning Publications 2003

[Leh01] M. M. Lehman, J. F. Ramil and G. Kahen: *A Paradigm for the Behavioural Modelling of Software Processes using System Dynamics*, Technical Report, Imperial College London 2001/8, September 2001

[Lou07] Neil Loughran, Pablo Sánchez, Nadia Gámez, Alessandro Garcia, Lidia Fuentes, Christa Schwanninger, Jasna Kovacevic: *Survey on State-of-the-Art in Product Line Architecture Design*, AMPLE Deliverable D2.1. Mar 30, 2007

[MA05] J. McAffer, J.-M. Lemieux: *Eclipse Rich Client Platform. Designing, Coding, and Packaging Java Applications*, Addison Wesley, 2005

[Kov07] Jasna Kovacevic, Mauricio Aférez, Uirá Kulesza, Ana Moreira, João Araújo, Vasco Amaral, Vander Ramos Alves, Awais Rashid, Ruzanna Chitchyan: *Survey of the state-of-the-art in Requirements Engineering for Software Product Line and Model-Driven Requirements Engineering*, AMPLE Deliverable D1.1. Mar 30, 2007

[Poh07] Christoph Pohl, Andreas Rummler, Vaidas Gasiunas, Neil Loughran, Hugo Arboleda, Fabricio de Alexandria Fernandes, Jacques Noyé, Angel Núñez, Robin Passama, Jean-Claude Royer, Mario Südholt: *Survey of existing implementation techniques with respect to their support for the requirements identified in M3.2*, AMPLE Deliverable D3.1. Jul 31, 2007

[Stu05] J. Stumpe and J. Orb: *SAP Exchange Infrastructure*, SAP Press., 2005