Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2008 Proceedings

Americas Conference on Information Systems (AMCIS)

2008

Language-Critical Enterprise and Software Engineering

Erich Ortner
Technische Universitat Darmstadt, ortner@winf.tu-darmstadt.de

Language-critical Enterprise and Software Engineering

Erich Ortner
Technische Universität Darmstadt, Development of Application Systems
ortner@winf.tu-darmstadt.de

ABSTRACT

With service-oriented architectures and the process-centric development of application systems, we are experiencing the beginning of a reorientation in how information technology is used in the global economy as well as in our private lives. Language-critical Organization Theory has thus become a discipline in research and teaching. Because of the global nature of these endeavors, it is important to look for a common basis for possible cooperation partners in this field. The necessary efforts are enormous and can only be managed successfully by working together. Looking at how the philosophy of science has developed throughout the twentieth century seems to imply that a) the primarily American analytic philosophy of science paired with b) the methodical constructivism of predominantly German origin (Erlangen and Constance) could form this common basis. In this paper, both philosophies, well-grounded in the works of Wittgenstein (1889-1951) and Frege (1848-1925), will form the foundation (Lorenzen, 1994) substantiating practical development in the field of e.g. service-oriented architecture. The common foundation will also disclose what people must be able to do or, respectively, what they must know (understand) if they want to work successfully in the IT-industry in the future.

Keywords

Enterprise Modeling, Language-Critical Development of Application Systems, IT-Management in Enterprises.

INTRODUCTION

This paper has been written against the background of the (practical) academic fields of "Mechanical Engineering", "Industrial Engineering" and "Applied Computer Science". Though, further interdisciplinary orientation in this case would prove to be beneficial. The mentioned academic fields together with the analytical philosophy of science (see e.g. Quine, 1960) and the methodical constructivism (see e.g. Lorenzen, 1968) constituted the background, when the language-critical approach on the development of application systems was first published as book (Wedekind and Ortner, 1980). This paper presents the further development of this approach (figure 1) to language-critical Enterprise Engineering.

The past 27 years have provided the insight that any noteworthy advances that support the further development of the language-critical enterprise engineering approach stem primarily from research in the field of Business Administration (e.g. Scheer, 1984) rather than German Business Informatics. Today, almost the entire Business Informatics community follows a non-language-critical path. About 15 years ago, a majority of members of the German Business Informatics community verifiably agreed on ignoring the language-critical approach (Wedekind and Ortner, 1980) in their work and even the work of their students and research assistants. If not ignored, the approach has been denounced as irrelevant or incomprehensible.

Language-critical Software Engineering (Ortner, 1993) has attracted more attention in "pure" (theoretical) Computer Science, which is however not willing to sustain the extension of the approach by language-critical Enterprise Engineering (Heinemann and Ortner, 2007).

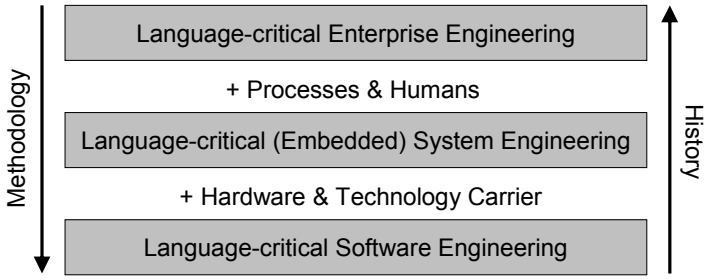

Figure 1. Historical versus Methodological Development of Application Systems

PROCESS-CENTRIC DEVELOPMENT OF APPLICATION SYSTEMS

The iterative orchestration of application systems (see figure 2) is one particular feature of service-oriented architectures in the following aspects:

- optimized work processes,
- ideal assignment of employees, and
- dynamic use of information technology using services.

Services, or more precisely service schemas, are application software that implements work procedures. They are developed on the basis of components and specified as algorithms.

Rather than having oboes, violins or triangles at ones disposal, the orchestration of application systems (see figure 2) makes use of human beings, organizational structures and technology. Whereby anyone who specifies e.g. organizational processes in the same way as computer processes and does not distinguish between human-related symbol processing and computer-based symbol processing (Oberweis and Broy, 2007), is not suitable for the development of process-centric application systems. Such a person lacks the interdisciplinary knowledge taught by some of the forward-looking chairs of Applied Computing and Application Systems at universities worldwide today.

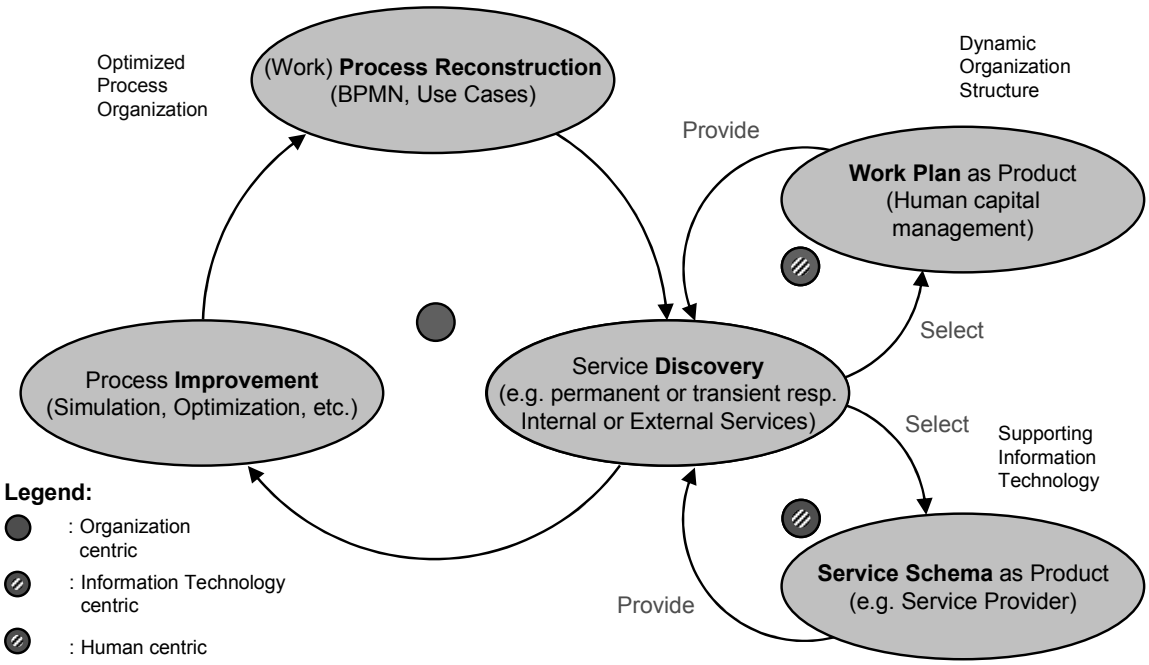

Figure 2. Iteration Paths of Orchestration: Organization - Technology - Man

INTERDISCIPLINARY LANGUAGE-CRITICAL SPECIFICATION OF IT-USE

Which skills and what kind of knowledge do developers, i.e. "business architects," "application developers," "solution architects," and so on, need for process-centric application development in order to successfully participate in projects of this kind or even execute such a project on their own? In his book "Der Flug der Eule" (The flight of the owl), Mittelstraß (1989) gives us an answer that is as clearly defined as it is simple:

"Anyone […] who has not studied interdisciplinary cannot perform interdisciplinary research."

The acquisition of interdisciplinary schemas and the understanding of them is a prerequisite for interdisciplinarity. Anyone who has only studied how to apply something will not be able to develop process-centric application systems. The following is a simple example that illustrates the profound understanding of the development process. The example reconstructs a data schema in Applied Computer Science.

Schematize the following sentence in object-language,

a) "Smith is a customer who is willing to pay."

by means of computer sciences, specifically the meta-language "relational model":

b) "Relation Name (key attribute(s); non-key attributes)"

in an interdisciplinary way, i.e. by different disciplines simultaneously. In order to accomplish this, a developer must not solely understand the sentence in object-language a) with respect to its business-driven generalization (schematization, norm), but additionally, the user must have agreed on the norm derived from it. In our example, this would mean that a society (language community) tolerates the following object-language norm:

a') "If we identify a person as a customer, we are allowed to characterize the customer more closely by the attribute 'payment behavior'."

Furthermore, it is necessary to realize or understand that the "relational model" is merely a different grammar (meta-schema) for representing the standardized (schematized) object-language "content". It is our goal to maintain customer data efficiently on the computer. Through modeling, we achieve the significant result of our interdisciplinary schematization:

b') "Customer (name; payment behavior)".

Interdisciplinary schematization (modeling) is one of the core tasks in Applied Computer Science such as Business Informatics. For the acquisition of interdisciplinary knowledge, e.g. in university courses of study, there is even a so-called "methodical order" (see figure 1). We can formulate it as follows, whereby the figures in brackets indicate the "sequence", i.e., the methodical order. Today specified processes are means for ends.

Computer Science:	*form (4) follows function (3)*
Business Informatics:	*applications (3) follow processes (2)*
Business and Social Sciences:	*means (2) follow ends (1)*

Theoretically, the methodical order, or course can be avoided. However, in practice, it is recommended to adhere to it. It is most advisable to "put on the socks before putting on the shoes", although, at least in theory, it may be possible to consider the reversed order. The problem in some of the programs of study in Computing Sciences is that interdisciplinary knowledge is not taught – even at the recently appointed German superior universities, an apparent lack in IT-architects has lead to the fact that students in bachelor programs of study merely concern themselves with pure computer sciences, i.e. (3) and (4). For those students who have not entered a practical profession by then, the Masters program of study will "ensure that they are acquainted with matters of Applied Computer Science" (Oberweis and Broy, 2007). Well, it is conceivable that disciplines become extinct!

Organization Modeling

For successful organization modeling (Enterprise Engineering) – especially with respect to optimization – differentiation is vitally important. Figure 3 illustrates the possibilities regarding work processes and structures.

The capability to differentiate clearly is critical to the ability to optimize. This is important for the object-language level, the application field, as well as for the meta-language level, the diagram language field. On both levels, the point is the reconstruction of connector words (e.g. to do) and topic words (e.g. to work). On the meta-language level, the developer gets

to know the modeling method in greater detail. On the object-language level, the grammar of the modeling language plays a vital role. In the latter case, the organizational expert knowledge of the relevant application domain must be represented in a structured way.

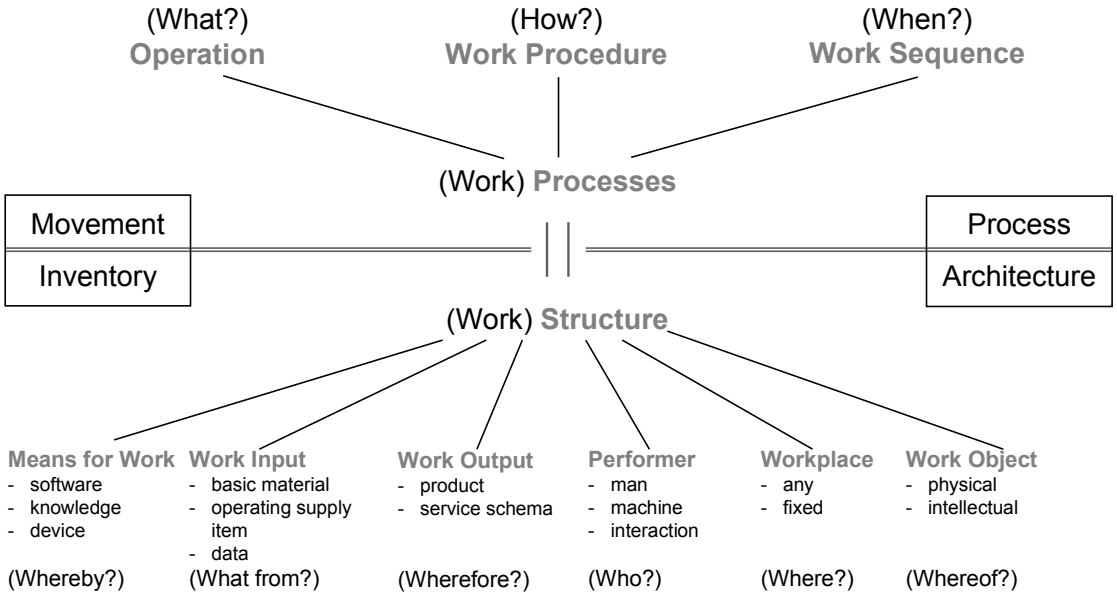

Figure 3. Differentiated Work Organization

Modeling (topic words of the application field) and structuring (connector words of the modeling language) are different but they supplement each other as complementary parts.

Even before the object-oriented system design, diagram languages have proved to be suitable modeling languages for the organization of a company's structures and processes. Use cases for example are especially useful for the structural aspect (see figure 4), while the Business Process Modeling Notation (BPMN) is suited ideally for the procedural aspect (see figure 5).

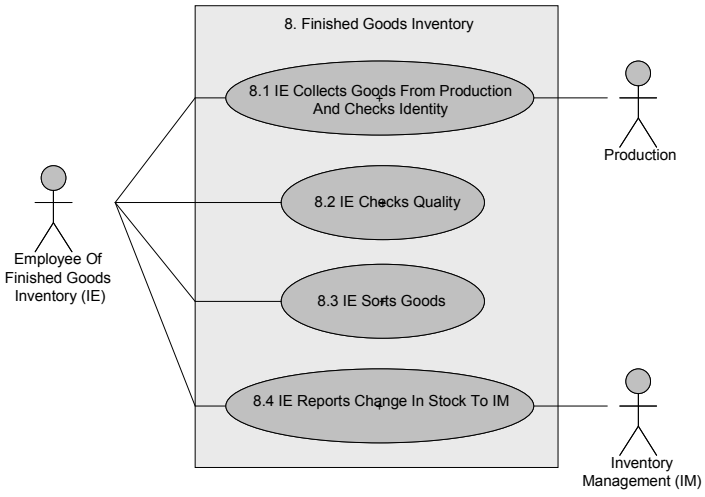

Figure 4. Use Case Diagram of Finished Goods Inventory

Detailed descriptions of modeling languages can be found in various case collections (Cockburn, 2001) or OMG manuals. However, anyone who later, in the system design, intends to specify the flow of work processes in greater detail is well advised to distinguish the aspects like "operations", "work procedures" and "sequence of work" or "workflows" orthogonally. The same applies to the organization structure and aspects such as "workplace", "vacancy", "employee" or "work material" (see figure 3). The optimization can now be considered sensibly and from different angles (aspects).

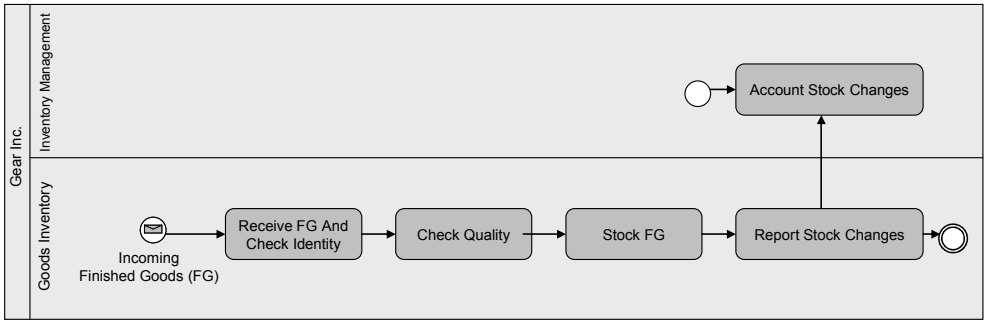

Figure 5. BPMN Diagram of Incoming Finished Goods

Method-Neutral Knowledge Reconstruction

The method-neutral knowledge reconstruction (Ortner, 1997) is primarily communicative and hardly any diagram representations are used.

In order to get a first picture of the important tasks, which are performed in close cooperation with the users, we classify them roughly in the following three parts:

- Collection of propositions that are relevant for development by talking to the users.
- Clarification and reconstruction of the expert terminology that has been used.
- Establishment of a common enterprise expert language.

The collection of propositions relevant to the development can be done by using a model, as shown in figure 6.

The model is intended to aid the collection and to ensure that all potential types of results for the system design have been scrutinized in consideration of their underlying expert knowledge. The following list contains several propositions that can be assigned to the above fields (see figure 6):

a) An account has an account number.
b) An account is opened.
c) Opening an account results in an opening balance.
d) The total of all debit line items must be the same as the total of all credit line items in double entry accounting.
e) A (personal) account is assigned to a business partner.
f) Shipment of goods is related to posting business transactions.
g) At the end of an accounting period, all of the accounts are closed, their values are entered in a profit and loss account and ultimately gathered in the balance.

In the clarification and reconstruction of the identified expert terminology the following "defects" are discussed and examined thoroughly with the future users or the company's experts.

Checking ***synonyms***

Check for words with the same meaning (extension and intension) that can be interchanged.
e.g.: MEMBER and ASSOCIATE have the same meaning for DATEV.
(DATEV is a computer center and software house for the German-speaking tax profession where the author worked as executive manager in software development for seven years.)

Eliminating ***homonyms***

Check for words that are written or pronounced in the same way but have a different meaning.
e.g.: STALK, which can mean either part of a plant or to follow someone around

Identifying ***equipollences***

Different names are used for the same objects (extension) from different perspectives (intension).
e.g.: Goods or merchandise of a company is referred to as STOCK from a quantitative perspective and INVENTORY ACCOUNT from a value perspective.

*Clarifying **vagueness***

As there is no clear delimitation (definition) of the terms in regard to their content (intension), it may not be clear which objects belong to each term (scope, extension)
e.g.: Does RESIDENCE, the place where a CONSULTANT works, belong to the term CHAMBERS for DATEV or not?

*Replacing **wrong designators***

Discrepancies between the actual meaning of a word and the meaning assumed at first (intension and extension)
e.g.: For DATEV, the CONSULTANT NUMBER does not define the function of a tax CONSULTANT, but it defines the USER RIGHTS a tax CONSULTANT has within DATEV.

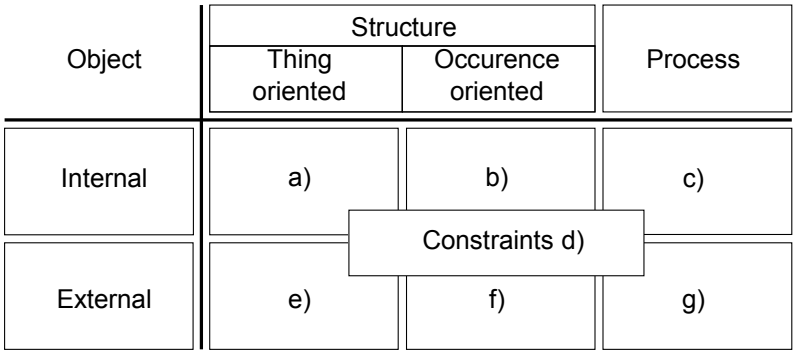

Figure 6. Classification Schema for Propositions about an Enterprise

This clarification results in further propositions relevant for development. Their relevance for the result types (system design) can be examined with the help of a classification schema (see figure 6). Work on building a common expert language for a company, which is aimed at integrating all of a company's knowledge resources, can be organized in different ways.

1. With the help of a repository, a kind of glossary will be created and administered. This glossary will contain all the terms that are important for an organization (language community), and should be designed for internal and external use.
2. A much more complex way, in comparison to (1.), of representing a company's knowledge is with an encyclopaedia. The encyclopaedia amounts to a conceptual schema for data but will go substantially further in respect to terminological coherences. This approach will distinguish inward and outward knowledge, which will be administered in a repository as an enterprise knowledge base.
3. The enterprise expert language is a rational interim language that is implemented on a meta-meta language level in the repository (Ortner, 1999). It is used for integrating and translating other languages used in a company. For users, it is not necessary to know the interim language itself.

Currently, the three variants discussed above can be found in industry worldwide. Vendor-independent research is done in the field of SOA under the catchword Enterprise Application Integration (EAI). Furthermore, companies like Oracle look into Application Integration Architecture (AIA) and offer products such as Fusion. Other vendors offer products like WebSphere (IBM) or NetWeaver (SAP) for the integration task.

General Object-Oriented System Design

After the development-relevant knowledge (see figure 6) has been reconstructed neutral to specific methods and technology (e.g. according to Ortner, 1997), and integrated into the overall knowledge base of an enterprise using common language, then, in the system design, this knowledge is transformed into the result types of an object-oriented solution to the task. Figure 8 shows an object-oriented software design according to Ortner and Schienmann (Ortner and Schienmann, 1996) that has been extended for the design of service-oriented architectures of an enterprise (figure 7).

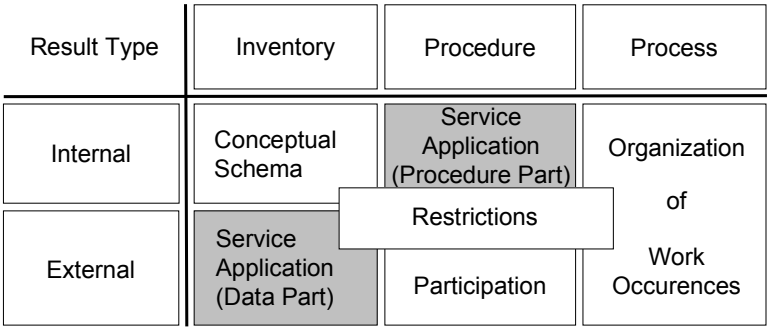

Figure 7. Extended Object-oriented Enterprise Design

When we speak of entirely object-oriented development of application systems, the enlightening step is the introduction of objects from computing sciences as grammatical objects. Grammatical objects are target points of language actions (e.g. writing, speaking, thinking) in a sentence, whereby they can also be replaced by pronouns in the sentence (e.g. one, he, him, this one). At school we have learned to speak of direct and indirect objects, genitive objects and various prepositional objects.

In contradiction to what many computer scientists still believe, when modeling and programming in computer science we do not concern ourselves with concrete or "ontological objects" such as this chair, that apple or my laptop. When speaking of objects "informatically" (i.e. modeling and programming), it is of particular importance that abstract types such as "class" in a repository or the term "invoice" in an application, are to be thought of as target points. The concrete, "ontological objects" are usually found in the application fields.

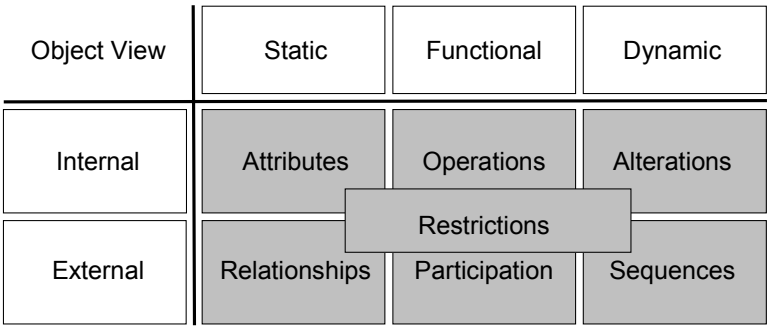

Figure 8. Object-oriented Software Design

Computer science is the science where students learn how to talk constructively about language (grammatical) objects, or more precisely, about abstract objects. Needless to say, we can still start from the concrete objects of the application fields in "Requirements Engineering" and when introducing the implemented solution, we can refer back to the users' concrete (ontological) objects.

The object-oriented approach in the development of application systems goes back to Platon. Platon classifies objects from the perspective of human beings and their languages into things (nouns, proper names) and actions, which can also be considered occurrences (verbs). If we transfer this classification to operating with data on a computer, the object-orientation (resp. its object) will be classified into the fields of data orientation (things) and procedure orientation (occurrences). This classification shows why object-orientation is universal. It encompasses data orientation (data classes) as well as procedure orientation (procedural classes).

Based on the results of organization modeling (enterprise engineering) and (embedded) system design (see figure 1), the results from figure 7 are modeled (enterprise engineering) in the following methodical order:

1. *Process modeling:*
   - BPMN diagrams (from organization modeling)
   - State machine diagrams
   - Activity diagrams
   - ...
   - Constraints (e.g. in Object Constraint Language (OCL))

2. *Participation modeling:*
   - Use cases (from organization modeling)
   - Sequence diagrams
   - Job descriptions (in the sense of structural organization)
   - ...
   - Constraints (e.g. organizational standards such as signature regulations)
3. *Procedure modeling:*
   - Class diagrams (data classes and procedural classes)
   - State machine diagrams
   - Activity diagrams
   - ...
   - Constraints (e.g. plausibility checks at data entry in service-oriented applications)
4. *Inventory modeling:*
   - Object type diagrams (for the conceptual schema)
   - Dataflow diagrams (for specification of data that are exchanged)
   - External schemas (extended as data classes)
   - ...
   - Constraints (e.g. semantic integrity rules for DBMS-enforced integrity)

Enterprise Engineering and the reconstruction of development-relevant expert knowledge from the application fields are highly communicative processes. Here, users and developers communicate very “intensively” (in great detail and clearly) with each other. Diagram languages play a minor role in this context. In contrast, the (entirely) object-oriented design of a SOA with diagram languages must be performed in an already highly "significative" way. This means that the terms of the object language and the meta-language (e.g. ”invoice” as an object-language terminus and “procedural class” as a meta-language terminus) should be displayed as independent as possible from their use in the judgment-context. The focus is on disclosing the types (software, concepts) that shall be implemented later. Diagrams are ideally suited for this purpose.

The diagram languages for procedure modeling are of course very similar to the diagram languages for process modeling. Procedure modeling comprises of the process parts (algorithms), which run as service-oriented applications while a process is being executed, as well as of those process parts, which can be specified in less detail since they involve human work (e.g. following work plans).

The order (1.-4.) chosen here serves merely as a recommendation. The modeling process is an iterative process, as every well-educated developer will know from practical projects (see figure 2).

CONCRETION IN THE LARGE

Process-centric development of application systems has been derived from data-centric (Wedekind and Ortner, 1980) development. The development paradigm “applications follow processes”, which is valid in today’s service-oriented architectures, complements, but does not replace the data-centric approach. Therefore, SOA stands for a new paradigm, not a shift in paradigm. The data-centric approach remains as important as ever, but due to the triumphant progress of object-orientation and component-based development, it is integrated in the overall architecture and work processes in a more “intelligent” way (Platon was right!). In addition to data processing, work organization (enterprise engineering) has become a subject in applied computer science (software engineering).

Figure 9 illustrates an enterprise that is organized as an application system in a process-centric way, considering the expert field (domain) and the logical structure (architecture).

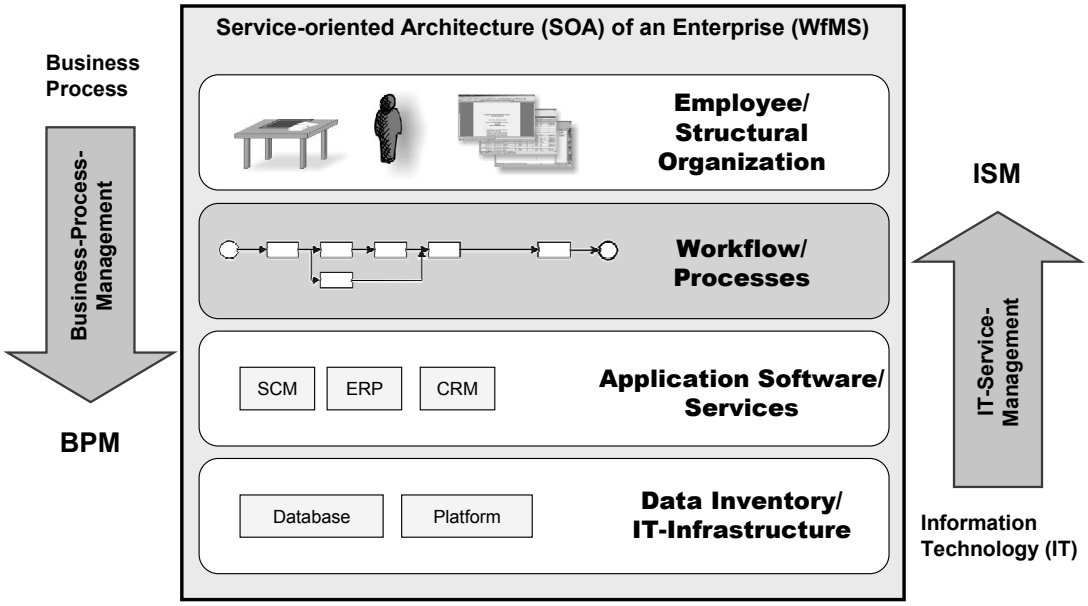

Figure 9. 4-Tier-Architecture of an Enterprise (SOA or WfM-Application)

Figure 10 shows an enterprise represented by software in an entirely object-oriented way. On the right, implementation aspects can be found; the right side lists the specific details of the information technology available for implementation today. We are therefore only talking about a "concretion in the large". "IT and solution architects", "integration developers" and "deployment managers" must be able to deliver this concretion for the enterprises (domains) that use information technology (Jablonski, Petrov, Meiler and Mayer, 2004).

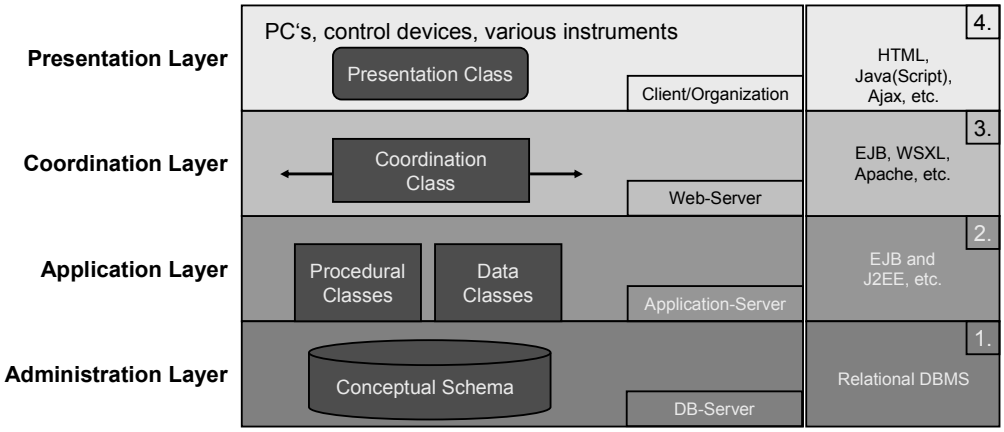

Figure 10. Entirely Object-oriented Concretion of SOA – The Software-Part

"Code development", which is of course also important, in particular from the point of view of the "service and solution testers" on site, is currently done in so-called "low-wage countries" by well-trained people (near and offshoring). In complementation to a "concretion in the large", we are now talking about a "concretion in the small."

DYNAMIC SUPPORT AND OPTIMIZATION OF WORK PROCESSES

For the dynamic management of application systems (see figure 9), it is necessary to create and use a meta-information system whose most important part is the repository system (Ortner, 1999) as for example described by (Berbner, Grollius, Repp, Heckmann, Ortner, and Steinmetz, 2007). In accordance to the much-noted work "The Quest for Resilience" by Hamel and Välikangas (Hamel and Välikangas, 2003), future enterprise networks will be implemented as elastic ecosystems (Corallo, Passiante and Principe, 2007) built from components of different categories. These systems must be assembled in the best possible way, thereby facilitating the systems to respond, possibly even self-actingly, to changing situations.

The ever changing job design (e.g. due to product changes) and work organization is crucial to this approach. This is done considering

a) the aspects: optimized processes, best possible employee assignment and dynamic IT-support (e.g. IT-services), as well as
b) the fact that some of the jobs that are part of these processes, are performed by employees who come from everywhere, or respectively, the jobs are done where personnel is available at low cost.

In this regard, organizing potential assignments for employees in work plans and establishing a global workflow management system (see figure 11) is exceedingly relevant. Such a system allows neutral (assignment-free) storage and maintenance of work processes. It could contain the IT-services (data and program schemas) that are used anywhere in the world as program-technological means (to work plans) for work in those processes. This way, an enterprise's IT-department organizes and controls the company's work processes worldwide in division of labor and dynamically using the Internet.

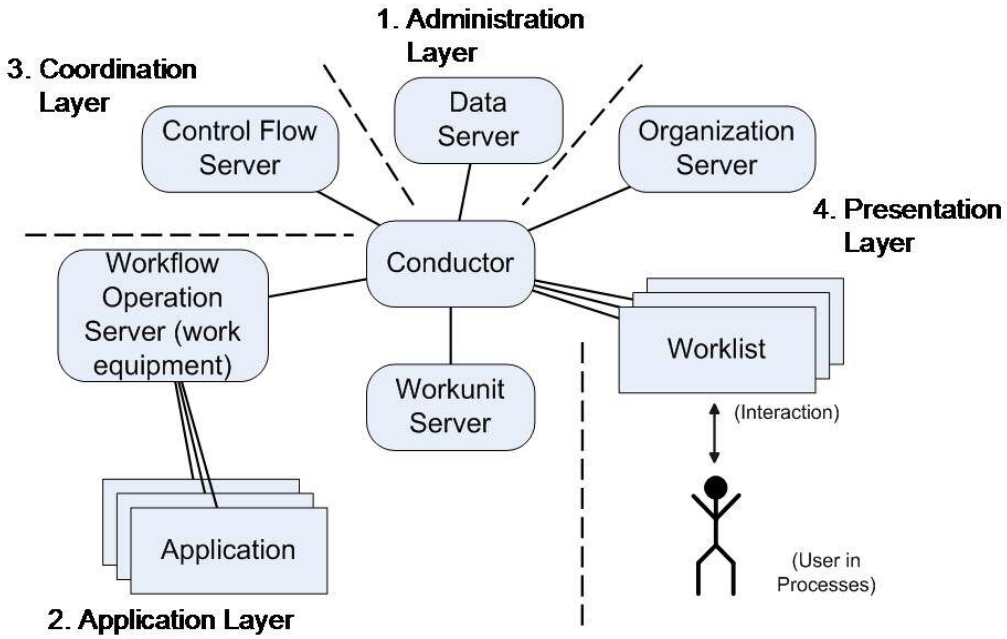

Figure 11. World Wide Workflow Management System (Jablonski, Petrov, Meiler and Mayer, 2005)

The protruding innovation of SOA is the extension of the concept of application systems by work and organizational processes of enterprises. This makes organization theory as it is found in Business and Social Sciences, the Engineering disciplines or in Enterprise Engineering an "integral", that is an interdisciplinary part, of Applied Computer Science. And this in a way that has not been enough seen in previous years.

Concepts and institutions like the German REFA-Association for Work Design or Methods-Time-Measurement (MTM) founded in 1924 (These are systems for time allotment that have been used in Sweden as of 1950, in Switzerland since 1957 and in Germany since 1960) suddenly constitute a field of activity and provide IT-businesses and enterprises worldwide with the knowledge that information and computing scientists possess. Due to Ubiquitous Computing, however, this also affects the courses of study of Enterprise Engineering, Business and Social Sciences, Mechanical Engineering, Electrical Engineering or Civil Engineering, as all of them are concerned with work science and process organization.

From the perspective of an employee, there are three possibilities to be considered when setting out to optimize work processes using IT:

1. to reduce people's workload through *automation* (resource: "software")
2. to *support* human work as for example using interactive applications (resources: "software" and "knowledge"), or
3. to improve people's work *qualifications* (resource: "knowledge")

Industrialization and automation were so successful in the previous decades that it is very advisable to revert our efforts with respect to the listing above. With globalization in mind as well as taking into consideration our worldwide division of labor, we should "invest much more in education and as little as possible in further automation efforts." Technological progress cannot be stopped, but a world which is becoming increasingly compact, can only cope with progress, if it is flanked by human education.

OUTLOOK

In Applied Computer Science, from a global standpoint, process-centric software and enterprise development constitute a new paradigm, but do not result in a paradigm shift. Managing data and managing processes are complementary and lead to entirely new job descriptions. In the expert languages of globally interacting IT-enterprises these new professions are called:

- IT-Architect
- Business Analyst
- Application Developer
- Service and Solution Tester
- Software Developer
- Deployment Manager
- Integration Developer
- Solution Architect
- Code Developer
- etc.

Nevertheless, people, who perform these jobs throughout the world, have the least say in *who performs which kind of work when and where*.

There is nothing more important for our survival than that the humanities take up the challenge to newly enter in a process of enlightenment. *Logic*, *Mathematics*, *Linguistics* and *Computer Science*, for example, are studies of the humanities. "Normative Logic and Ethics" (Lorenzen, 1984) as well as their advancement to an "Encyclopedia Philosophy and Philosophy of Science" (Mittelstraß, 1996) provide us with the necessary *fundamental education and terminology*, in the sense of a *Universal Literacy*, to fulfill this task.

Therefore, we appeal for language-critical computer science (Mittelstraß, 1996) to become basic education for all citizens. As a matter of course, this basic education should be graded and differentiated into interdisciplinary (rather universities) and infradisciplinary (rather schools) knowledge.

The root of the matter is teaching a disciplined and reflected use of language.

A great difficulty of our profession, however, lies in the inability to become language-critical overnight. Being language-critical is the consequence of a long and committed process of reflection and self-observation while using language. Despite this difficulty there does not exist a more effective way for attaining (work-) process centric application systems solutions than the language-critical way.

Therefore, the challenge is to provide curricula which help to shorten and level, at early stages, the path for young people to a language-critical life in times of mobile communication and the participation of all in the boundless offers of the World Wide Web.

REFERENCES

1. Berbner, R., Grollius, T., Repp, N., Heckmann, O., Ortner, E. and Steinmetz, R. (2007) Management of Service-oriented Architecture (SoA)-based Application Systems, *Enterprise Modelling and Information Systems Architectures*, 2, 1, 14-25.

2. Cockburn, A. (2001) Writing Effective Use Cases, Addison-Wesley, Boston.
3. Corallo, A., Passiante, G. and Prencipe, A. (2007) Digital Business Ecosystems, Edward Elgar Publishing, Cheltenham.
4. Ghani, H., Koll, C., Kunz, C., Sahin, T. and Yalcin, A. (2007) *Concept for the BITKOM University Challenge 2007* "*Best Process Architecture",* http://www.bitkom.org.
5. Hamel, G. and Välikangas, L. (2003) The Quest for Resilience, *Harvard Business Review*, Sept. 2003.
6. Heinemann, E. and Ortner, E. (2007) Memorandum zum Verhältnis von System- und Anwendungsinformatik, http://www.metainformationen.de/downloads/unternehmensingenieur/MemorandumZumVerhltnisVonSystemUndAnwendungsinformatik.pdf, 24.04.2008.
7. Jablonski, S., Petrov, I., Meiler, Ch. and Mayer, U. (2005) Guide to Web Application and Platform Architectures, Springer, Berlin.
8. Lehmann, F.R. (1999) Fachlicher Entwurf von Workflow-Management-Anwendungen, B.G. Teubner Verlagsgesellschaft, Stuttgart.
9. Lorenzen, P. (1968) Normative Logic and Ethics, B.I.-Wissenschaftsverlag, Zurich.
10. Lorenzen, P. (1994) Konstruktivismus, *Journal for General Philosophy of Science*, 25, 1, 125-133.
11. Mittelstraß, J. (1989) Der Flug der Eule – Von der Vernunft der Wissenschaft und der Aufgabe der Philosophie, Suhrkamp Verlag, Frankfurt.
12. Mittelstraß, J. (ed.) (1996) Enzyklopädie Philosophie und Wissenschaftstheorie, J.B. Metzler Verlag, vol. 1 (1980), vol. 2 (1984), vol. 3 (1995), vol. 4 (1996), Stuttgart.
13. Nussbaum, D., Ortner, E., Scheele, S. and Sternhuber, S. (2007) Discussion of the Interaction Concept focusing on Application Systems, *Proceedings of the IEEE International Conference on Web Intelligence 2007*. In press.
14. Oberweis, A. and Broy, M. (2007) Informatiker disputieren über Anwendungsnähe der Disziplinen, *Computer Zeitung*, Monday, 16.07.2007.
15. Ortner, E. (1997) Methodenneutraler Fachentwurf – Zu den Grundlagen einer anwendungsorientierten Informatik, B.G. Teubner Verlagsgesellschaft, Stuttgart.
16. Ortner, E. (1999) Repository Systeme, Teil 1: Mehrstufigkeit und Entwicklungsumgebung, Repository Systeme, Teil 2: Aufbau und Betrieb eines Entwicklungsrepositoriums, *Informatik-Spektrum*, 22, 4, 235-251 resp. 22, 9, 351-363.
17. Ortner, E. and Schienmann, B. (1996) Normative Language Approach – A Framework for Understanding, *Proceedings of the 15th International Conference on Conceptual Modeling,* Cottbus, Germany, October 7-10, Springer, Berlin, 261-276.
18. Quine, W.v.O. (1960) Word and Object, The MIT Press, Cambridge.
19. Scheer, A.-W. (1990) EDV-orientierte Betriebswirtschaftslehre, 4th ed., Springer, Berlin.
20. Wedekind, H. and Ortner, E. (1980). Systematisches Konstruieren von Datenbankanwendungen – Zur Methodologie der Angewandten Informatik, Carl Hanser Verlag, Munich.