ENTERPRISE SYSTEMS CONFIGURATION AS AN INFORMATION LOGISTICS PROCESS
A Study

Mats Apelkrans and Anne Håkansson
Jönköping International Business School, Dept of Informatics, Jönköping, Sweden
Dept of Information Science, Computer Science, Kyrkogårdsgatan 10, Uppsala, Sweden

Keywords: Visualisation, Knowledge modelling, Information Logistics, Enterprise Systems, Configuration, Unified Modeling Language, Verification and Validation

Abstract: In this paper, we suggest using rule-based descriptions of customer's requirements for Enterprise Systems implementing Information Logistics. The rules are developed from the users' requirements and inserted as schedules to the Enterprise System. The output, from testing these rules, is a list of modules and parameter settings to configure the system. By using rules, we can, at least partly, automate the configuration process by traversing the several modules and thousands parameters that are in an Enterprise System. From the list, we can select the modules and the parameters that meet the customer's requirements. Then these selected modules and parameters are visually presented through a kind of Unified Modeling Language diagrams, to support the user investigation and then to configure the system either manually or automatically. Every attempt to match a customer's requirement to the contents of the knowledge base within the Enterprise system can be thought of as an Information Logistics Process. The output from such a process must be examined by the user, which can give rise to a new call to the Information Logistics process. In other words the configuration work is done through a dialogue between the customer and the knowledge base of the Enterprise system.

1 INTRODUCTION

One of the largest problems in Business Informatics is to configure chosen Enterprise System (ES) software to meet a customer's specific requirements. This can be seen as a kind of customization of the ES software. However, following Summer (2005) and Hedman and Kalling (2002) we will call it configuration. According to Hedman and Kalling (2002) *"the actual process of implementing an ERP system is complex. Every ERP system has to be configured. Configuration involves adapting the generic functionality of the package to meet the needs of a particular organization (usually by setting parameters in tables), such as what calendar should be applied for a firm with locations in several countries. All in all, the configuration might affect thousands of configuration tables and can take years to complete."* Methods and discussions of configuring enterprise systems are also found in (Keller and Teufel 1998; Olson 2003). Summer (2005) talks about Configuration Management. Configuration Management provides product specific configuration support for companies that must build products for their customers. Technology vendors, appliance vendors, and computer vendors are examples of companies, which need to create product configurations, and make price quotes. A software package can contain a number of modules each with a great number of parameters needing to be set for optimal values. The problems are how to find them and how to give them the best possible values. The configuration cannot 100% meet the requirements, because some modules can be cancelled and there can be functionalities missing in the package. In that case, one may have to add extra software. This will give rise to another problem: integration of this added software to the original ES package.

The configuration efforts are both time-consuming and costly (Adam and Sammon, 2004). There is also a lack of human competence. People working with ES configuration usually have only a

Apelkrans M. and Håkansson A. (2007).
ENTERPRISE SYSTEMS CONFIGURATION AS AN INFORMATION LOGISTICS PROCESS - A Study.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 212-220
DOI: 10.5220/0002370602120220

partial knowledge of an ES packages' many parameters.

In this paper, we concentrate on how to find the modules and the parameters from the customers' requirements on the enterprise system. We present a study on how to partly automate the configuration process using rule-based descriptions of customer's requirements and the modules of the ES package. These rule-based descriptions are stored in a Knowledge Base.

The use of production rules on top of the content of the ES is necessary to keep track of the order of the tasks performed by the system. The rules handle all the modules and the parameters used in systems. With rules, for example, it can be assured that the calculation of total price in not performed before having the information about both the price per product and the quantity of that product. The rules in the system's knowledge base already have the order needed for different tasks but the customers' requirements may not meet these. Therefore, we match the rules of the knowledge base with the rules developed from the customers' requirements. The matching between requirements and ES package will be performed as an Information Logistic Process (ILP). Hence, there will be a number of ILP calls. This is the outer loop. The user, through a dialogue, controls the operation of this loop.

In section 2 and 3 we define the information logistics process including three phases of the process. In section 4, we show the common modules and parameters and in section 5 we present rules from users' requirments. The last section includes matching the users' requirements to the system's rule description, the result of applying rules and verification of the knowledge base.

2 INFORMATION LOGISTICS PROCESS

In this paper, we discuss Information Logistics from an information logistics process perspective: An information logistics process transforms a given input into some form of output. The input is some kind of fragmented information or knowledge description, which derives from a so-called information supplier. This input information can be handled either manually or automatically by the system. The process output is an information product that will be made accessible and delivered to the information receiver who will use the information. This workflow is called the Information Logistics Process (ILP). Input to the ILP in our study is the customer requirements. From these the ILP communicates with the knowledge base to produce a list of modules and parameters but also marking failures, see Figure 1.

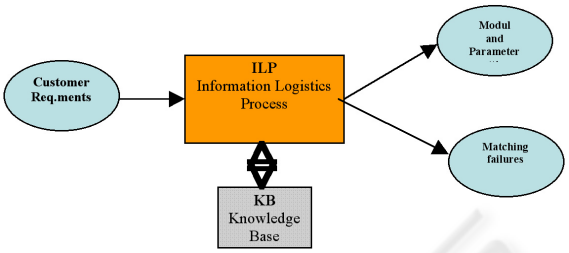

Figure 1: The Information Logistics Process (ILP)

The ILP processes are implemented with different methods found in the computer science area. Simple ILP processes are just straightforward database solutions; other need real-time machinery in order to deliver the information product at right time to the right place. Real-time components can handle time management and communication details to facilitate the distribution of information to right place in right time. (Frauenhofer Institute, 2006). Still others need knowledge management. A question is what can possibly be automated and what will still be contained in the dialogue between users of ILP and the ILP process. The ILP processes have to handle knowledge, or more properly, expressions of knowledge descriptions. This paper focuses on the possible use of a knowledge base to partly automate a process.

3 THE THREE PHASES OF THE INFORMATION LOGISTICS PROCESS

In Apelkrans and Håkansson (2005) we provided rules for e-invoicing between enterprise systems as an ILP process. In that case a company handled the process as a third part Information Logistics Provider (3ILP). A closer look at the ILP process points out three different phases:

- The first phase is for information supply (IS), the information needed to trigger the next information production phase. As an example a customer sends invoice data to 3ILP and in our study the input to ILP will be a customer's requirements presented with data and rules.
- The second phase handles information production (IP). In the 3ILP example IP will produce correct information for the receiver's ES and in the

study of this paper IP is searching for the correct ES modules and the parameters that are affected. The results are handed over to the last phase for distribution.
The third phase is for information distribution (ID). In the 3ILP case the desired receiving ES are connected and information is delivered. In our work the desired output is put together from the ILP process, i.e. suggested modules and parameters are presented to the user. These three phases are presented in Figure 2.

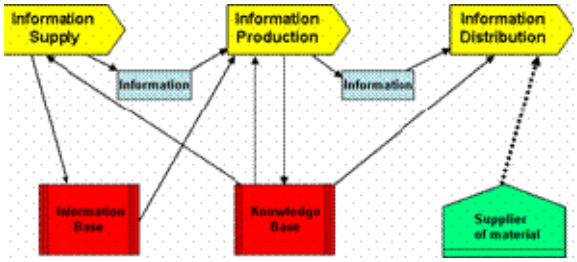

Figure 2: The Information Logistics Process three phases.

The reason why ID is completed with a possibility to use hard material can be illustrated in the 3ILP case with the fact that some receivers of invoice can't take it electronic so we must produce a paper variant and send by ordinary mail.

As mentioned earlier, all phases in ILP need to handle knowledge descriptions in some way. Such knowledge descriptions need to be specified from case to case. A general question is: how should this knowledge be elicited, stored and maintained? Another question is: is it possible to automate knowledge management in some way?

The IP phase is of course the main one of those three and can be further divided into three phases.

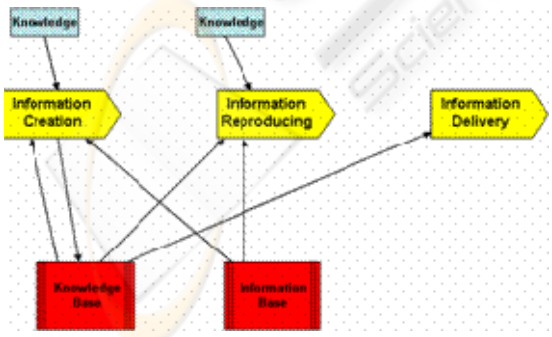

Figure 3: A proposal for an IP architecture

One phase for creating information, another for reproducing already created information, and finally a phase leveraging the produced information to the Information Distribution (ID) sub-process. Figure 3 below gives a suggestion of the architecture of the Information Production (IP) sub-process.

A simple example of the IP sub phases is handling a CD request containing a number of tracks. The request needs be produced once and if another customer wants the same CD, the request simply has to be reproduced with little or no cost and completed with information to be sent to the right receiver.

In order to handle both information and predefined rules for managing the information, i.e., a kind of normative knowledge descriptions, the architecture contains an Information Base (i.e., database) and a knowledge base. In the Information Base one will find actual and/or stored information and the knowledge base contains rules for their use. In our study, most work (which is a matching rule system) is done in the Information Creation sub-process.

Our method contains an outer loop, which means that ILP is called several times in order to refine the module and parameter selections. The method also contains an inner loop for comparing the customer's requirements with the contents of the system.

4 COMMON MODULES AND PARAMETERS

An ES is a standard software package usually packaged in a number of modules fitting different application areas. As an example we present the following list of modules:

General Ledger
Fixed Assets
Sales & Receivables
Relationship Management
Service Management
Purchase & Payables
Inventory
Manufacturing
Capital Requirements Planning
Human resources

These modules are possible to configure, i.e., set certain parameters, to omit certain parts of a module and possibly adding some extra software. The behaviour of an ES module depends on the values permitted to those parameters, which are decided by the user. As an example of parameters, we give the following list for the invoice function, see Table1:

Parameter	Possible values
Costing Method	Standard, FIFO LIFO Specific Average
Price/Profit Calculation	Profit = Price-cost, Price = cost + profit
Sales Units:	piece, 10, dozen, 100
Unit cost: Number of decimals	0,1,2,3
VAT calculations:	%-age groups
Product type	Material, part, complete product

Table 1: Parameter types with possible values

As mentioned above, our proposed method has an outer loop, a dialog between the user and the ILP knowledge base. From user requirements, written in rule form, this loop describes in more detail the modules needed, the appropriate parameter list and its parameter values. Finally the loop will come up with a suggestion for software in the ES modules. This software might be redundant since it is difficult to find unused code in the system. Moreover, this unused code might be needed later on. An open question is still how to present the outcome to a given ES.

The knowledge base in our ILP process contains rule descriptions of every module there is in an ES. In many cases a function in a module needs to be detailed into sub-functions. These sub-functions are also expressed by rules.

The inner loop of our method compares a customer's requirements rule description with those in the knowledge base in order to find matching and mismatching between them.

The ES modules are graphically described by ARIS diagrams (Scheer et al, 2002) like the one shown in Figure 5. Those descriptions are fairly complicated and need to be structured in graphs with sub-graphs. We call them functions with sub-functions or processes with sub-processes. How to handle them in a rule-based system is covered in Section 5.

5 BUILDING RULES FROM REQUIREMENTS

Customers have requirements on the Enterprise Systems to adjust it to the organization. To meet a customer's specific requirement, the ES has to be configured. Configuration aims at choosing appropriate modules, refining the modules but also setting their parameters to optimal values.

As mentioned before, the customer's requirements are originally presented in graphical forms (using the ARIS diagrams suggested by Scheer et al, 2002), which are designed as schedules. In the schedules, the customers are specifying the parts they need in a module that best suits their business by developing, changing and extending the schedules in the ES. We use these graphical specifications to develop rules for the module.

Since these software packages can contain many modules with thousands of parameters we have to handle the requirements in an efficient way. Firstly, a user has to study the ES Basic Terminology List (for the list see e.g. Navision Attain 2002) in order to avoid mismatches caused by the wrong terminology. This is needed since the current system does not detect the use of wrong terminology. Secondly, the user must insert the parts of the schedules of the processes used by the system and make these as complete as possible. The parts used in the system are triggers, functions, application system types and organizational units. Each part has its form and colour to distinguish them from one another. The different parts are presented in the figure below, Figure 4.

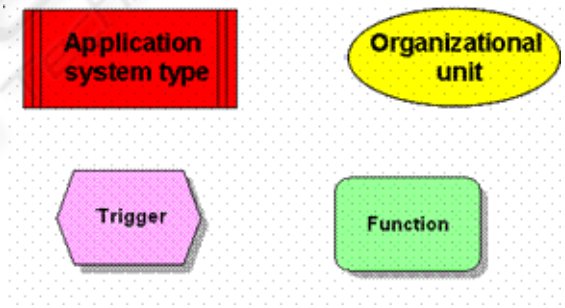

Figure 4: Different parts used in a schedule

The triggers are the events (e.g. "Start price calculation") handled by the system, which either are inputs to functions or outcomes ("Result from price calculation") from functions. The functions, on the other hand, are the actions provided by the system (e.g. the actions needed to calculate the correct price). These functions are some tasks to be performed by the system. As mentioned above, some of the functions have to be detailed in a sub-function. The application system types can symbolize different things but in our case they describe a temporary storing of information. The information is stored in a database to be used later on in the graph. The organization units point out the

responsibilities of a certain action. Some of the actions require some job to be carried out within the company.

For expressing the parts in the software system, we have chosen antecedent-consequent rules, i.e., production rules to represent the requirements. The rules allow us to automate the configuration process by using rule-based descriptions directly onto customer's requirement. The rules can be applied on customer's requirements because of the nature of the requirements. These requirements use parts that can be expressed as rules, like the logic operations, triggers and actions (also called functions). The logic operation, with "and" and "or", can be applied in rules by letting the "and" be captured within the rules and the "or" be captured by using several different rules. Moreover, the triggers are applied as facts and actions as rules.

The logic interpretation of the requirements is the execution order of the rules. The order is the relationships between the different rules. For example, if one rule comprises another rule, both have to be interpreted and completed before both are usable in the conclusion. Moreover, if the rule comprises two different rules, R2 and R3 in that order, the rule R2 will be tested before rule R3. The order those rules are applied in the rule becomes the order of the execution.

The output from these rules, so called conclusions, is a list of modules and parameters. This list is used for configuration of the enterprise system by comparing the list with the contents of the system also expressed in rules. This list is checked for correctness of the modules and parameters, manually, and then for configuration the system's modules and parameters, automatically.

5.1 Example of Rules Built from Users' Requirements

The customer's requirements for the Enterprise System are drawn as a schedule. This schedule contains the triggers, functions, application system types, organization units, logic operators and parameters needed by the module to accomplish a task. However, for the rules some of these parts are omitted. The rules handle the triggers, functions, logic operators and parameters. The organization units are the units that take care of the product at the company and do not need to be expressed in the rules. Neither does the application system types need to be expressed because it is fetches and sends data to and from the database.

An example of building rules from the customer's requirements is using the schedule for the invoice and production process, see Figure 5. This process can be described with a couple of rules. For example, applying rules on top of this schedule can be done by the use of only one rule, but for the quality of the knowledge base and following the schedule, we use several rules.

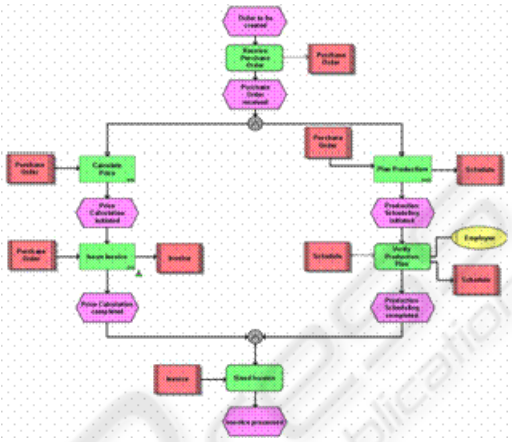

Figure 5: An example of invoice and production process (For details see appendix.)

There will be an overall rule that works as an all-embracing rule, which invokes the facts and the rules needed to reach a conclusion. In this example, the overall rule starts by invoking the rule "`Purchased order`" and then calls for several rules, i.e., "`Price calculation`", "`Production scheduling`" and "`Send invoice`" to reach the conclusion "`Invoice processed`". The conclusion to the rule "`Purchased order`" is "`Purchased order received`" and to the rule "`Price calculation`" "`Price calculation completed`". We also have the conclusions "`Production scheduling completed`" and "`Invoice Sent`".

The system uses both pre-stored facts, i.e., stored in the database, and user-given facts, i.e., facts inserted by the users during consultation with the system. Hence, the facts, which correspond to the triggers, are found by searching in the database. If these facts are not found, the system asks the user. In this example, the system is launched by the rule "`Process Invoice`" which calls the rule "`Purchased order`". Then the latter rule invokes the question "`Order to be created`" which answer or answers (if several answer alternatives) became the user-given fact. Thus, the answer becomes a fact that is initiating the consultation. The fact "`Price calculation initiated`", on the other hand, may be a pre-stored fact because the data can be stored in the database of the system. Then, this trigger only checks whether or not the session

has all data to be able to continue the consultation of the system.

Besides invoking the fact "Order to be created", the rule "Purchased order" call rule "Receive purchase order", which can have a schedule of its own that is the sub-functions mentioned above. Thus, every function may use a schedule like the one presented in Figure 5. This schedule also needs to be executed before continuing in the consultation.

Following the schedule in Figure 5, the rule "Price calculation" invokes the rule "Calculate price", the fact "Price calculation initiated" and the rule "Issue invoice". The rule "Production scheduling" invokes the rule "Plan production", fact "Production scheduling initiated" and the rule" Verify production plan". Finally, the rule "Send invoice" is checked. All the rules in this example are not specified, e.g., the last rule, which either use a fact or a schedule. Even though it is not pointed out in this paper, it illustrates the complexity of rules that are produced even by small examples.

The corresponding knowledge base, to the rules presented above, is expressed in a kind of logic syntax:

```
Rule "Process Invoice"->
 Rule "Purchased order" and
 Rule "Price calculation" and
 Rule "Production scheduling" and
 Rule "Send invoice".

Rule "Purchased order"->
 Fact "Order to be created" and
 Rule "Receive purchase order".

Rule "Price calculation"->
 Rule "Calculate price" and
 Fact "Price calculation initiated" and
 Rule "Issue invoice".

Rule "Production scheduling" ->
 Rule "Plan production" and
 Fact "Production scheduling initiated"
and
 Rule "Verify production plan".

Rule "Send invoice".
```

Once these rules have been developed and established into the system, they will become the customer's requirement rules.

5.1.1 Visualising the Developed Rules

Expressed in the syntax of a programming language these rules together with the

relationships can be difficult to handle. To support the users in understanding the rules, the rules and relationships can be illustrated visually. As visual tool, we use diagrams similar to the ones in UML (Booch et al., 1999; Jacobsson et al., 1999). Some of these diagrams have proved to be useful for rules modeling the Information Logistic, as shown in Apelkrans, and Håkansson, 2005, and modelling for acquiring knowledge in Håkansson, 2001.
One might argue to use the schedules as presented in Figure 5 but this does not illustrate the execution order of the rules. Therefore, we will use a UML-alike sequence diagram to present the rules described above, see Figure 6.

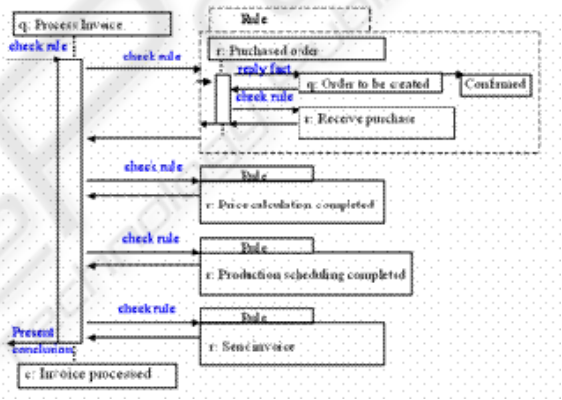

Figure 6: A visual presentation of the rules in the example of invoice and production process

In the figure the overall rule "Process Invoice" is presented. It begins with following the lifeline by invoking the related rules. In this case, the execution is starting with "Purchased order" illustrated as a square with broken lines. This rule explores the fact "Order to be created" and the rule "Receive purchase order". After checking these rules, the overall rule continues with the rule "Price calculation" presented in a so-called package. In this diagram, the package icon is used to denote that there are contents in the rule and to see it, the package has to be unfolded.

There are also two other rules in packages, the " Production scheduling" rule and the "Send invoice" rule. At the end of the lifeline, the conclusion is presented, i.e., for this schedule it is "Invoice processed".

The functions, or actions, sometime have to be detailed in a sub-function. These sub-functions are also rules, which are hidden in the package. To examine the package contents, the package is unfolded. Unfolding the rule "`Price calculation`" is to display the contents of that rule, see Figure 7.

Figure 7: A visual presentation of the content of a package

Unfolding the rule package supports the customer checking the contents. In this example, it is found that one rule consists of two other rules "`Calculate price`" and "`Issue invoice`" and a fact `Fact "Price calculation initiated"`. This folding/ unfolding feature is necessary to make the rules comprehensible for the customer. Too much information makes it hard to get an overview of the rule-base whereas packages hide important information.

6 MATCHING REQUIREMENT RULES TO THE KNOWLEDGE BASE RULES

The users build the schedules through drawing and it is not certain that they will remember all the parts that have to be in the schedule. For instance, in the example presented above, Figure 5, the users may have omitted some important parts, for instance, the parameters.

The rules developed on the requirement schedule have to be tested for viability and usability in the system. The system must have a full collection of full-fledge modules that contain all the parts and parameters used in the system. These modules are implemented as rules stored in the knowledge base. These rules are used as templates to which the developed rules, also called requirement rules, are tested. For each part that is omitted in the requirement rules, the system needs to ask whether the users purposely omitted the parts or accidentally.

There might be another mismatch between the rules in the knowledge base and requirement rules do to the use of terminology used by the ES and implemented in the rules in the knowledge base. The system must find these mismatches, both from checking the rules in the knowledge base as presented above, but also from the requirement rules. If parameters are omitted, the system asks for answers on these. In the invoice case, it is to ask for the costing method, price, profit calculation, sales unit, unit costs, VAT and product type. Each of these become facts in the rules. If a rule is wrong, the user is asked to check it. The user either needs to change the rule or reduce it. Moreover, if the rules are in the wrong order, the user is asked to change the order of the rules. It should be a possibility to override these as well but then that enterprise system may not be guaranteed to work properly.

If there are parts in the requirement rules that are not present in the rules of the knowledge base, the system asks the intension of these parts the users have specified. The users are asked to check the rule and its content but also check the terminology.

6.1 The Result of Applying Rules

The result of matching all the knowledge base rules against the requirement rules is a list with the modules and parameters (with chosen parameter values) the users have requested. This list is presented to the customer or user. The user should carefully study the behaviour of the ES to be able to suggest the right content of the modules. The user might not be satisfied with the list and the modules and/or the parameters need to be changed to undergo the matching process again. This will be iterated until the user is satisfied with the list. However, if the user agrees to the list there are two options. Either the list is directly applied to the ES or printed as a report of what should be manually adjusted in the system.

6.2 Verifying the Knowledge Base

The Enterprise system's ability to expand the rule base with the requirement rules for modules and parameters depends on the ability to assure correctness and completeness of the knowledge.

Therefore, the system should handle verification of the knowledge base. Verification is the demonstration of software consistency, completeness and correctness at each stage. However, in this work we only work with the consistency.

Developing and updating the rule base involves, among other things, checking the consistency. In rule-based systems the rule base is consistent if for each interpretation all facts are true (Beauvieux, 1990). Consistency checking includes testing whether the system produces similar answers to similar questions (Polat and Guvenir, 1993). Consistency problems that can occur are conflicting, redundant, subsumed and circular rules (Polat and Guvenir, 1993). A conflicting rule is when it contains a fact that is both true and false at the same time. Redundancy occurs when several premises, that are identical, are included in a rule. A subsumed rule means that two rules produce the same result but one is more restrictive than the other. Circular rules is when a rule has dependencies to other rules that prevent the rules to reach any conclusions. Those rules have premises that use each other. Depending on the schedules of the user, the system must check so it will not run into verification problem. This is an automatic test and should be run for every schedule. If there is a cross-reference between several schedules, these are tested together.

7 CONCLUSIONS

In this paper, we have presented a rule-based description of the customer's requirements on an Enterprise System. Input to the system is the form of requirement schedules and output is the modules and parameters to be used by the Enterprise system. The rules built from the requirements are matched to the rules in the knowledge base. The missing data between these rules will be asked for.

This study is restricted to a few examples of handling an ILP process configuration of a customer chosen ES. Therefore, there are a number of rules and situations for several different ES that need to be examined further. This might require adjustments of the rule structures or the modules used in the specific ES.

Future research will focus on managing the outer loop, automating the process of transferring user requirements given in graphical form to rule systems. We will also build in the terminology for the system. In the current version, the user has to study the ES Basic Terminology List in order to avoid mismatching by wrong terminology. In the coming version, the system will present a list with terminology used by the system from which the user can use the right word. Since the number of terms is vast, this list will be long so the user will either choose from the list or write the words directly in the interface of the system. If the user misspells the word or uses wrong terminology, the system will suggest the words that are commonly used in the context for the modules.

Configuration costs a lot of money today, and sometimes does not give a satisfying solution. On the other hand, our proposal will also cost a lot of money to fully implement the time-consuming effort to build up the Enterprise systems Knowledge Base. Each specific ES needs its own Knowledge Base completed with rules for modules, actions and parameters. Furthermore, a drawback with our suggestion can be that graphical descriptions may not familiar to ES users and therefore difficult to use.

REFERENCES

Adam, F. and Sammon, D., 2004. The Enterprise Resource Planning Decade: Lessons Learned and Issues for the Future. • *Idea Group Publishing.*

Apelkrans, A. and Håkansson, A., 2005. Visual knowledge modeling of an Information Logistics Process - A case study. *ICICKM-2005, 2nd International Conference on Intellectual Capital, Knowledge Management and Organisational Learning* Dubai, Förenta Arab Emiraten.

Beauvieux D. 1990. A general consistency (checking and restoring) engine for Knowledge base. *Proceedings from ECAI-90.* Pitman Publishing, p. 77-82.

Booch, G. Rumbaugh, J., and Jaconson, I. 1999. The Unified Modeling Language User Guide. Addison Wesley Longman, Inc.

Buck-Emden, R., 2000. The SAP/R3 Systems: An introduction to ERP and business software technology. *Addison-Wesley*

Frauenhofer Institute 2006 *www.isst.fhg.de/englisch/download/34868_I-Log-4-Seiter-engl-2.pdf*, 10 december 2006.

Hedman, J. and Kalling, T., 2002. IT and Business Models. *Liber*

Håkansson A., 2001. UML as an approach to Modelling Knowledge in Rule-based Systems. *ES2001 The Twenty-first SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence.* Peterhouse College, Cambridge, UK; December 10th-12th, 2001.

Jacobson, I. Booch, G. and Rumbaugh, J. 1999. The Unified Software Development Process. Addision Wesley, USA.

Keller, G. & Teufel, T., 1998. SAP R/3. Process Oriented Implementation, *Addison-Wesley.*

Navision Attain Terminology handbook, 2002
Olson, D. L, 2003. Manegerial issues of ERP Systems.,*McGraw-Hill*
Polat F. and Guvenir H.A 1993. UVT: A Unification-Based tool for Knowledge Base Verification. *IEEE Expert* , June, No. 3.
Scheer, A-W., Abolhassan, F., Jost, W. and Kirchmer, M. [eds] 2002. Business Process Excellence. ARIS in Practice. *Berlin: Springer Verlag*.
Summer, M., 2005. Enterprise Resource Planning. *Prentice Hall.*

APPENDIX

Figure 5 in full scale