This is a postprint of the paper Jeusfeld, M.A., P. Backlund, and J. Ralyté (2007): Classifying Interoperability Problems for a Method Chunk Repository. Proceedings of the 3rd International Conference Interoperability for Enterprise Software and Applications, Funchal, Portugal, March 28-30, 2007. Originally published by Springer, DOI: 10.1007/978-1-84628-858-6_35

Classifying Interoperability Problems for a Method Chunk Repository

Manfred A. Jeusfeld[1], Per Backlund[2], Jolita Ralyté[3]

[1] Tilburg University, CRISM/Infolab, 5000 LE Tilburg, The Netherlands
Manfred.Jeusfeld@uvt.nl

[2] University of Skövde, P.O. Box 408, SE 541 28 Skövde, Sweden
Per.Backlund@his.se

[3] CUI, University of Geneva, Rue de Général Dufour, 24, CH-1211 Genève 4, Switzerland
Jolita.Ralyte@cui.unige.ch

Keywords: interoperability problem classification, method chunk, method chunk repository

1 Introduction

The Software Engineering Body of Knowledge (Swebok, 2004), notably a book with 200 pages mentions the word interoperability just twice, once as an example for a system requirement and the second time as the title of a standard for library data models. This stands in contrast to the challenges of globalizing economy that demands solutions for an exploding number of interoperability problems (Interop, 2006). So, is interoperability just a sort of user requirement that will emerge from the system implementation if the system developers are just careful in implementing them? We claim that this is not true because interoperability is not just achieved by a technical implementation but by addressing interoperability problems at all stages of the interaction between multiple partners, i.e. both in the business domain and in the ICT domain.

As part of the INTEROP initiative (Interop, 2006), we aim at designing a repository that stores solution descriptions for interoperability problems. In earlier papers (Ralyté et al., 2006; (Backlund et al., 2006), we have reported on how to describe solutions. We proposed the concept of method chunks originally developed for situational method engineering. In this paper, we focus on how to

represent interoperability problems as exposed by application cases, i.e. situations in which interoperability problems occur.

The vast array of interoperability problems calls for a domain-dependent knowledge management approach, which takes technical as well as business and organisational matters into account. Successful solutions to interoperability problems may then be stored in the form of method chunks as proposed in Situational Method Engineering (Kumar and Welke; 1992). Project-specific methods may then be created by selecting and assembling method chunks (Ralyté and Rolland, 2001b; Mirbel and Ralyté, 2006) stored in a method repository (Brinkkemper et al, 1998; Firesmith and Henderson-Sellers, 2001; Mirbel and Ralyté, 2006). The knowledge base provided by the repository is useful when dealing with problems pertaining to the interoperability domain. In particular, we find it useful in the early stages of a project. Instead of providing one universal method our approach aims to provide a knowledge base of reusable method chunks, which can be composed to form a project specific method.

In the remainder, we first introduce the concept of a method chunk repository (MCR) and a meta-case tool for situational method engineering for interoperability (MCTI). A meta-model is developed that links method chunks to application cases via the explicit concept of *interoperability problem*. This meta-model represents the structure of the method chunk repository. Afterwards, we derive from example cases the classifiers for interoperability problems. A problem classifier is a kind of descriptor that relates an interoperability problem to the context in which it occurs, e.g. the life cycle phase in which it occurred. In the last section we provide guidelines for applying this classifier in characterising method chunks and identifying interoperability problems in application cases.

2 Method Chunk Repository for Interoperability

The problem of enterprise interoperability is very complex. It not only concerns software and technologies but also enterprise knowledge and business references that must be shared. In order to achieve meaningful interoperation between enterprises, interoperability must be achieved on all layers of an enterprise which means that a multitude of interoperability problems and opportunities have to be resolved and designed. We claim that it is impossible to create one universal method supporting all possible interoperability issues. Moreover, we are convinced that the future of Systems Engineering will not see just one approach but a multitude of approaches depending on the type of system and the degree of reuse of solutions. Future systems will range from global data collection, analysis and presentation to dynamic systems for mass-customised product design. We therefore propose to adopt the ideas of Situational Method Engineering (Kumar and Welke, 1992) which promotes the notion of reusable method component also called method fragment (Brinkkemper et al., 1998) or method chunk (Ralyté and Rolland, 2001a, Mirbel and Ralyté, 2006) and the selection and assembly of these components according to the situation of the project at hand (Brinkkemper et al., 1998; Ralyté and Rolland, 2001b).

In this work we propose situational method engineering as a means for encoding situated knowledge about achieving interoperability in the form of method chunks each of them addressing one or more specific interoperability problems. A repository-based tool has to be defined in order to support method chunks storage, indexation and retrieval. We call this tool the Method Chunk Repository (MCR). The MCR becomes really useful if it is included into a collaborative meta-case tool providing services for method chunks engineering as well as for the selected method chunks enactment in a specific interoperability case. In the following sub-sections we present these three notions namely method chunk, MCR and collaborative meta-case tool for interoperability.

2.1 Method Chunk

We use the definition of a method chunk provided in (Ralyté & Rolland, 2001; Mirbel & Ralyté, 2006) and adapted to the interoperability domain in (Ralyté et al., 2006). This latest method chunk metamodel allows to link best practices for achieving interoperability to specific interoperability problems. It covers best practices from the business domain (e.g. aligning the business processes of enterprises) as well as from the ICT domain (e.g. integrating heterogeneous product catalogues). The main role of a method chunk is to provide guidelines to the system engineer for realising some specific system development activity (i.e business modelling, requirements specification, design, etc.) as well as to provide definitions of concepts to be used in this activity. These two kinds of method knowledge, namely method process and product parts, are captured in the method chunk body. For example, the method chunk providing guidelines for integrating two business process models will also define the meta-model that the integrated business process model should correspond.

The descriptor part of a method chunk includes a set of attributes allowing to characterise the situation in which this method chunk is meaningful. A detailed classification of these criteria related to the information systems development in general, named Reuse Frame, is proposed in (Mirbel & Ralyté, 2006). This classification framework provides criteria related to the critical information systems development aspects such as organisational (i.e. contingency factors, project management aspects, system engineering activities), human (i.e. required expertise, level of involvement) and application domain (i.e. application type, level of legacy reuse, technology). But it does not explicitly include criteria specific to the enterprise interoperability domain. In our work we extend the Reuse Frame with our interoperability problems classifier presented in section 3 of this paper. That allows us to relate explicitly each method chunk to one or several interoperability problems.

The concept of a method chunk forms a complementary approach to using patterns as proposed in (Chen, 2005). Patterns may be stored in a method chunk repository. One advantage of using a ME approach is that patterns will be related to each other as well as to the type of interoperability problems they solve, which will facilitate their use.

2.2 Method Chunk Repository

The prerequisite for situational method engineering is a method repository containing a large collection of method chunks. Different propositions for method repositories are given in (Saeki et al., 1993; Van Slooten and Brinkkemper, 1993; Plihon et al., 1998; Ralyté, 1999; Firesmith and Henderson-Sellers 2001; Mirbel and Ralyté, 2006). All these works focus their attention on the structure, representation and storage of method chunks but do not really consider their evaluation and their suitability in different application cases.

In our MCR, besides method chunks, we aim to capitalise knowledge related to the experience and best practices of method chunks application in specific industrial cases. Therefore, as shown in Fig. 1, the MCR stores two kinds of knowledge: the reusable method chunks and the descriptions of their application cases including experience reports and evaluation how method chunks and method chunk assemblies fit in these cases. The application cases should also be characterised by using the interoperability classification framework. The contribution of such a practical method chunks applicability evaluation is multiple. It helps:

- To improve method chunks characterisation and to specify the situation in which the method chunk applicable more precisely;
- To rank method chunks providing solution to the same or similar problems;
- To extract new method chunks from experience reports
- To identify the most applicable method chunk assemblies and to store them in the MCR as new aggregate method chunks.

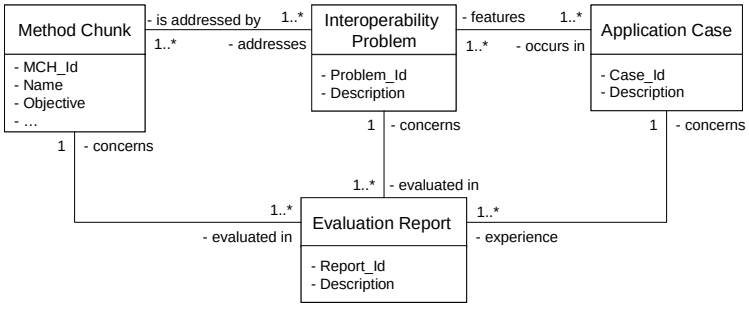

Fig. 1. The method chunk repository relates method chunks to their application cases via applicability evaluation repports.

By collecting method chunks in a MCR our approach provides accessibility for method users. This is an important feature of any knowledge repository (such as a method chunk repository or a pattern repository). A flexible classification scheme, such as we propose, addresses a number of issues concerning: tool support for creating method chunks and patterns, providing reliable techniques for access, storage, search and retrieval of knowledge as well as traceability. In particular, the evaluation reports and application cases provide information of successful

application of method chunks. Hence our approach forms a complement to the pattern approach proposed in (Chen, 2005) which omits connections between patterns.

2.3 Meta-case Tool for Interoperability

Situational method construction process asks for a specific software support named Computer Aided Method Engineering Tools (CAME). According to Harmsen et al., (1994) a CAME tool should provide support for the following method engineering activities: determination and valuation of contingency factors, storage of method chunks in a method base, retrieval and assembly of method chunks, validation and verification of the obtained situational method.

While there is now consensus on the functionality that a CAME tool should provide, considerable work has still to be done to achieve implementation meeting this functionality. A number of meta-CAISE products and prototypes such as Decamerone (Harmsen, 1995), MetaEdit+ (Kelly et al., 1996) and MViews (Grundy and Vanable, 1996) and Mentor (Si-said et al., 1996) have been developed which implements this functionality partially.

In this work we design a Meta-Case Tool for situational method engineering in the Interoperability domain (MCTI) including required method engineering features as well as method enactment and evaluation functionality as shown in Fig. 2 illustrating the boundary model of this tool.

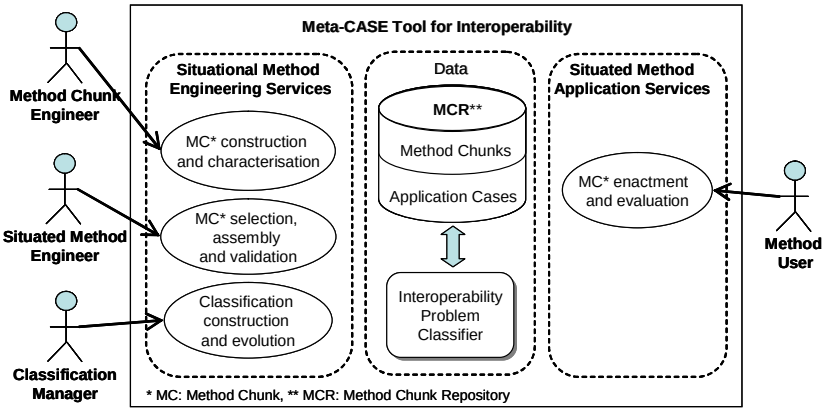

Fig. 2. Boundary model of the Meta-CASE Tool for Interoperability (MCTI)

As show in Fig. 2, we identify four main actors of the MCTI named method chunk engineer, situated method engineer, classification manager and method user. The first three actors use the MCTI for method engineering purpose while the last one is an application engineer which applies the method created for a particular application case. Table 1 summarises the goals of each actor.

Table 1. MCTI actors and their goals

Actor	Goal
Method chunk engineer	The goal of the method chunk engineer is to capture knowledge to specific interoperability problems as reusable method chunks that can be used in different application cases and to characterise method chunks following the classification scheme.
Situated method engineer	The goal of the situated method engineer is to find a set of method chunks that can be assembled into a coherent method that addressing a particular (interoperability) development/analysis need in a particular application case.
Classification manager	The goal of the classification manager is to develop and evolve interoperability classification schemes for classifying method chunks so that they are easy to search and navigate.
Method user	The goal of the method user is to be able easily and efficiently test/analyse/apply method chunks to specific cases, as well as describe experience of using these method chunks in his/her specific case.

The main use cases identified in the boundary model (Fig. 2) help us to identify services that the meta-case tool has to provide to the end-users. Besides, they also serve as a starting point for more detailed scenario descriptions of human-computer interaction and working environment of the end-users.

One potential way to achieve the desired functionalities is to extend commercially available modeling tools to also cover the needs for situational method engineering. This can be achieved by creating an extension of the tool that enables the representation of methods and method chunks in terms of meta-models. Several of the major modeling tools already have some form of repository support built in and many more tools can be integrated using technologies such as e.g. Netbeans (Netbeans, 2006).

3 Classifying Interoperability Problems

The classification framework has the purpose to associate method chunks as well as application cases to re-occurring interoperability problems. By tagging the method chunks with suitable instances of interoperability problems, we index the chunks much like books and articles are indexed in a library: the indexing is supporting the search for method chunks that address a certain interoperability problem. In the same way, actual cases are described in terms of the interoperability problems that are occurring in them. The challenge is to index problems and solutions in such a way that a match between the two is made possible.

3.1 Ontological Dimensions for Classifying Interoperability Problems

Interoperability problems are occurring in a certain situation within a project concerned with the interaction of multiple organizations and their information systems, hence covering both the business/organizational domain and the ICT domain. The following questions guide the definition of the classification framework:

1. From which knowledge domain can we draw expertise to understand the interoperability problem?
2. During which lifecycle stage does the problem occur?
3. Which types of products are involved in the observed interoperability problem?
4. Which types of processes were active when the problem occurred?
5. Which types of human or automated producers are involved in the problem?

The five questions are translated into five classification dimensions as follows.

Knowledge dimension. Iivari et al. (2004) propose five ontological domains (Type KnowledgeDomain in Fig. 3), which are based on a review of the state of the art in current IS research. These five domains cover the area of Information Systems well. The *organisational domain* refers to the knowledge about social contexts and processes in which the information system is used. The *application domain* refers to the knowledge about the application domain for which the information system is intended. The *IT application domain* refers to the knowledge about typical IT applications and their use in a certain application domain. The *technical domain* covers the hardware and software of an information system. In the technical and IT application domains we find issues of data management and software management, hence relating the IS field closely to the field of software engineering. Finally, the *development process knowledge* refers to the methods and tools used in systems development.

Lifecycle dimension. The lifecycle dimension characterises the phase in which some situation is observed or some activity can take place. At the highest level of granularity, we distinguish the four phases: (1) *business-strategic* – the phase of a project in which strategic business decisions are made, (2) *business-operational* – the phase in which business activities are executed, (3) *ict-development* – the phase in which some ICT solution is developed, and (4) *ict-execution* – the time when some ICT system is performing operations. This level can be further decomposed, for example the phase *ict-development.analysis* is the phase in which the specification of an ICT systems is analysed.

Product dimension. The product dimension specifies types of products that are relevant in some observed situation or that are involved in some activity. Possible values are: *model-type* – the involved products have the nature of models, *document-type*, *notation*, and *language*. Like before, specializations are formed like *model-type.data-model* or *model-type.source-code.java-program*. For documents, we suggest to form specialisations according to the structure of the document, e.g. d*ocument-type.contract.sla* for a service-level agreement.

Process dimension. The process dimension has to be distinguished from the lifecycle phase. It is defined as the processes that are active in some observable situation. At the highest level, we distinguish three kinds of processes: *human-process*, *automated-process*, and *human-computer-interaction*. At deeper levels,

processes like *human-process.meeting.group-modeling-session* are expressed. Another example is *automated-process.data-exchange*.

Producer dimension. Producers are human or automated actors hat are capable of creating and processing some products. For the purpose of interoperability problem classification, we distinguish *role* characterising the responsibilities of a human actor (e.g. *role.system-analyst*), *team* (e.g. *team.development-team*), and *system* (e.g. *system.tool.diagram-editor* or *system.enterprise-system.crm-system*). Note that producers are observable at any lifecycle stage.

The last four dimensions are adapted from the Open Process Framework (Henderson-Sellers, 2003).

3.2 Meta-model for Interoperability Problems Classification

Fig. 3 shows the addition of the problem classifier concept to the MCR meta-model. An interoperability problem is identified and described in terms of its symptom, for example "the systems of partner 1 and partner 2 cannot exchange data". Each interoperability problem can have multiple problem classifiers linking it into the business and ICT context, i.e. the universe of terms that stakeholders use when talking about interoperations of systems. The problem classifier (Fig. 3) provides a finer-grained scheme than the one utilized in (Chen et al., 2006). Therefore we expect the problem classifier to be of use when assessing interoperability problems.

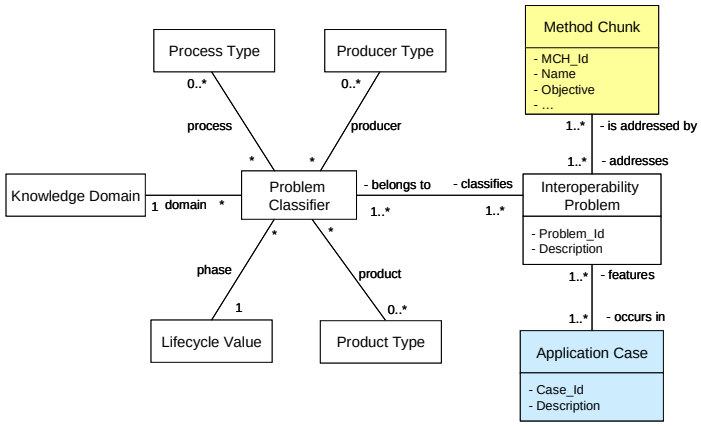

Fig. 3. Meta-model for interoperability problems classification

The problem classifiers assiciated to interoperability problems are standardised statements about the situation in which the interoperability problem has been observed or can be observed. The allowed tags are from a controlled ontology of keywords for interoperability (see section 3.1). Each individual problem classifier

is thus a viewpoint on the problem. The combination of all classifiers associated to the same interoperability problem characterises the problem in a comprehensive way. The restricted vocabulary for the five dimensions supports keyword-based search for method chunks and application cases but goes beyond it. The user can supply keywords from the five dimensions and the MCR shall respond by those interoperability problems whose problem classifiers match the supplied keywords. Since a problem classifier is a statement about an observation, it can be checked inhowfar it is true in the context of the user.

For example, an interoperability problem may have occurred in an application case where a cross-organisational team negotiated a contract about a cooperation involving linking the IT systems of the organization. Then, one problem classifier is given by the 5 values: process type=human-process.negotiation, product type=document-type.contract, knowledge domain=organisational-domain, lifecycle value=business-strategic, producer type=team.interorganisational-team. The users current situation could be that there is a problem with producing a contract between multiple partners. The match with the list of existing problems classifiers returns all situation in which the product type is a contract and the process type is negotiation. The user may then decide whether or not the returned problem classifier is true in her situation as well. The fact that multiple values are combined to a single expression is exactly the difference to simple keyword-based approaches where any combination of keywords may be expressed regardless whether they make sense or not. In contrast, our problem classifiers are true statements about interoperabilty problems as experienced in application cases and as successfully solved by method chunks.

4 Applying Interoperability Problems Classification

4.1 Characterising Method Chunks

Tagging of method chunks by interoperability problems is the responsibility of the author of the chunk, i.e. method chunk engineer. For standard chunks such as the reverse engineering of a conceptual data model out of a database schema, the author can create a suitable entry in the list of interoperability problems, e.g. ‘understand legacy databases’. In many cases, a method chunk will be the generalization of successful solution of a case problem. Then, the interoperability problem will have been stored in the MCR as result of classifying a case.

4.2 Assessing Case Situation

A case in the context of the MCR is a situation of a user (or group of users) that includes an interoperability problem that requires to be addressed in a structured way. The classification of the case problem is a manual process and is the first step of the method chunk selection and assembly service of the MCTI in order to conctruct a case-specific method. The classification limits the search space of applicable solutions, i.e. method chunks, as well as the type of change to be

expected from the solution. We suggest the following approach for the classification of the case problems:

1. Determine the IS domain of the case problem: The IS domain is characterising the type of knowledge that is necessary to understand the case problem. For example, dealing with heterogenous data structures belongs to the IS domain 'development-process'. Here, the Swebok (2004) knowledge base can be used to characterise the field.
2. Determine the lifecycle stage: Possible values are 'business-strategic' (specifying that the interoperability problem encountered is about the business domain and about a strategic decision to be taken by the business partners), 'business-operational', 'ict-development' and 'ict-execution'. For example, resolving heterogenous data structures would require analysis, modelling and implementation activities in the 'ict-development' stage according to method chunks, possibly stemming from an established software engineering methodology, in accordance with its specifics.
3. Determine the involved product types (if applicable). The example will involve implementaion on a specific platform in a specific language. The method chunks associated to the current problem will contain existing solutions previously classified to suit the situation.
4. Determine the involved process types (if applicable). A method chunk may be classified on the process dimension with respect to the to the human process of analysing the semantics of the current data stuctures in order to make them possible to match.
5. Determine the producer type (if applicable): stakeholders, involved organizations, team composition, tools used for production.
6. Determine the interoperability problem: The set of problems is build upon experience, i.e. whenever a case problem occurs one looks up whether a similar problem is already stored in the method chunk repository. The interoperability problems are the most specific abstractions of past case problems. Only the interoperability problems shall be associated to method chunks, i.e. their potential solutions.

This stepwise approach focuses the situated method engineer towards the most relevant interoperability issue for the case problem to be classified. The closer he/she describes the case problem along the five categories, the easier is the classification process. Furthermore, we associate experience reports of applying the chunks, which will provide the case classifier information. It will help in assessing the suitability of the method chunk in question.

5 Conclusion

The proposed approach will enhance knowledge management by means of using a method chunk repository to store reusable method chunks. The collection and storage of method chunks is supported by MCTI services for the creation, management, execution and evaluation of method chunks. In order to make

knowledge retrievable the method chunks have to be classified. This is done by classifying the interoperability problems by using the proposed problem classifier. The advantages of using the classifier are:

- It allows for more efficient retrieval of stored knowledge. This is an important feature for user services of a knowledge repository.
- It provides the possibility to use multiple classifiers makes it possible to provide a richer characterisation of method chunks
- The problem classifier is useful irrespective of how knowledge is stored, i.e. in the form of patterns or method chunks.
- A problem classifier is a meaningful statement about a situation, i.e. it is not just a combination of keywords but an expression about a past or future observation.
- The problem classifier augmends the characterisation of patters in terms of conceptual, technical and business barriers as proposed by (Chen et al., 2006).

The strength of the proposed scheme is the incorporation of organisational as well as business and technology aspects of interoperability. It also associates interoperability to existing bodies of knowledge within the information systems and software engineering domains. The proposed meta model can directly serve as the schema for an interoperability-aware method chunk repository. Prototypes based on the schema have been developed within the InterOp task group on method engineering and are currently evaluated.

References

Backlund P., Ralyté J., Jeusfeld M.A., Kühn H., Arni-Bloch N., Goossenaerts J.B.M. and Lillehagen F. (2006, in press) An Interoperability Classification Framework for Method Chunk Repositories. *In the Proceedings of the 15th International Conference on Information Systems Development (ISD 2006)*, Springer, pp. 697-710.

Brinkkemper S., Saeki, M. and Harmsen, F. (1998) Assembly Techniques for Method Engineering. *Proceedigns of the 10th Conference on Advanced Information Systems Engineering, CAiSE'98.* Springer, LNCS 1413, pp.381-400.

Chen D. (Ed.) (2005) Practices, principles and patterns for interoperability. *Deliverable D6.1 Interop Network of Excellence IST – 508011.* http://interop-noe.org/deliv/d6.1/ Accessed 2006-11-19.

Chen D., Dassisti M, and Elvesaeter B. (Ed.) (2006) Interoperability knowledge corpus Intermediate Report, *Deliverable DI1, Interop Network of Excellence IST – 508011.* http://interop-noe.org/deliv/DI1/ Accessed 2006-11-19.

Firesmith D. and Henderson-Sellers B. (2001) The OPEN Process Framework. An Introduction. *Addison-Wesley*.

Grundy J.C. and Venable J.R. (1996) Towards an Integrated Environment for Method Engineering. *In Chelenges and Strategies for Research in Systems Development. W.W. Cotterman and J.A. Senn (Eds.), John Wiley & Sons.* Chichester, pp.45-62.

Harmsen A.F., Brinkkemper S. and Oei H. (1994) Situational Method Engineering for Information System Projects. *In Olle T.W. and A.A. Verrijn Stuart (Eds.), Mathods and*

Associated Tools for the Information Systems Life Cycle, Proceedings of the IFIP WG8.1 Working Conference CRIS'94, pp. 169-194, North-Holland, Amsterdam.

Harmsen A.F. and Brinkkemper S. (1995) Design and implementation of a method base management system for situational CASE environment. *Proceedings of 2nd APSEC Conference, IEEE Computer Society Press*, pp 430-438.

Henderson-Sellers B. (2003) Method engineering for OO systems development. CACM 46(10), pp. 73-78.

Iivari J., Hirschheim R. and Klein H.K. (2004) Towards a distinctive body of knowledge for Information Systems experts: coding ISD process knowledge in two IS journals. *Information Systems Journal* (14) pp. 313-342.

INTEROP (2006) Interop Network of Excellence IST – 508011 Presentation of the Project. http://interop-noe.org/INTEROP/presentation Accessed 2006-11-18

Kelly S., Lyyttinen K. and Rossi M. (1996). Meta Edit+: A fully configurable, multi-user and multi-tool CASE and CAME environment. *Proceedings of the CAiSE'96 Conference, Heraklion, LNCS 1080, Springer Verlag*, Crete, Greece.

Kumar K. and Welke R.J. (1992) Method Engineering, A Proposal for Situation-specific Methodology Construction. *In Systems Analysis and Design: A Research Agenda, Cotterman and Senn (eds), Wile*y, pp.257-268.

Mirbel I. and Ralyté J. (2006) Situational Method Engineering: Combining Assembly-Based and Roadmap-Driven Approaches. *Requirements Engineering*, 11(1), pp. 58–78.

Netbeans (2006) Netbeans Metadata repository. [On-line]. Available at http://mdr.netbeans.org/ Accessed 2006-02-22.

Plihon V., Ralyté J., Benjamen A., Maiden N.A.M., Sutcliffe A., Dubois E., Heymans P. (1998) A Reuse-Oriented Approach for the Construction of Scenario Based Methods. *5th International Conference on Software Process (ICSP'98)*, Chicago, Illinois, USA.

Ralyté J. (1999) Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base. *Proc. of the 10th Int. Workshop on Database and Expert Systems Applications (DEXA'99), 1st Int. REP'99 Workshop*, Florence, Italy.

Ralyté J., Backlund P., Kühn H. and Jeusfeld M. A. (2006) Method Chunks for Interoperability. *Proceedings of 25th International Conference on Conceptual Modeling (ER2006). LNCS 4215, Springer*, pp. 339-353.

Ralyté J. and Rolland C. (2001a). An Approach for Method Reengineering. *Proceedings of the 20th International Conference on Conceptual Modeling (ER2001), LNCS 2224, Springer-Verlag*, pp.471-484.

Ralyté J. and Rolland C. (2001b) An Assembly Process Model for Method Engineering. *Proceedings of the 13th Conference on Advanced Information Systems Engineering (CAISE'01), LNCS 2068, Springer-Verlag*, pp. 267-283.

Saeki M., Iguchi K., Wen-yin K., Shinohara M. (1993) A meta-model for representing software specification & design methods. *Proceedings of the IFIP¨WG8.1 Conference on Information Systems Development Process, Elsevier Science Publishers B.V.* (North-Holand), pp 149-166.

Si-said S., Grosz G. and Rolland C. (1996) Mentor, A computer aided Requirements Engineering Environment. *Proceedings of the 8th International Conference on Advanced Information Systems Engineering (CAISE'96), LNCS 1080, Springer* .

van Slooten K. and Brinkkemper S. (1993) A Method Engineering Approach to Information Systems Development. P*roceedings of the IFIP¨WG8.1 Conference on Information Systems Development Process, Elsevier Science Publishers B.V.* (North-Holand), pp. 167 – 186.

SWEBOK (2004) Guide to the Software Engineering Body of Knowledge - 2004 Version. Available at http://www.swebok.org/. Accessed 2006-11-16.