A Model-Based Approach for the Management of Electronic Invoices

Rodrigo García-Carmona, Álvaro Navas, Félix Cuadrado, Boni García, Hugo A. Parada G., Juan C. Dueñas

Universidad Politécnica de Madrid
Departamento de Ingeniería de Sistemas Telemáticos
{rodrigo, anavas, fcuadrado, bgarcia, hparada, jcduenas}@dit.upm.es

Abstract. The globalized market pushes companies to expand their business boundaries to a whole new level. In order to efficiently support this environment, business transactions must be executed over the Internet. However, there are several factors complicating this process, such as the current state of electronic invoices. Electronic invoice adoption is not widespread because of the current format fragmentation originated by national regulations. In this paper we present an approach based on Model-Driven Engineering techniques and abstractions for supporting the core functions of invoice management systems. We compare our solution with the traditional implementations and try to analyze the advantages MDE can bring to this specific domain.

Keywords: Electronic invoice, Model-Driven Engineering, Model Transformation, Enterprise Applications.

1 Introduction

Electronic commerce is not something new. The buying and selling of products via electronic means have been in use for a long time now. However, there are still some aspects which hamper its massive adoption, such as the widespread use of electronic invoices. This way, not only the transactions could be automated, but also the financial, accounting and payment processes. Recent advances in aspects such as XML Security and Encryption have made those innovations possible. Nonetheless. most companies, especially small and medium-sized, are still adapting their financial processes for the use of electronic invoices.

Moreover, electronic invoice adoption presents additional problems because of even the smallest companies having to operate in a worldwide market: the myriad of standards existing in different countries for the management of invoices, governed by the national regulations. Therefore, the conversion from paper to digital format is being hindered by the need to support several standards and transform between them seamlessly.

To address this problem we propose an alternative to the traditional architecture for an invoice management system that employs the Model Driven Engineering paradigm. Instead of managing documents of specific formats, we propose to define a

model layer for increased abstraction, expressivity and extensibility. With this paradigm, we believe that development of an efficient and adaptable invoice management system could be done with much less effort. Even more, the system will be more robust, comprehensible and extensible.

The structure of the document is as follows: In the next section we will provide a brief summary of the current state of the electronic invoice, paying special attention to the particular situation of Spain. In section three we will describe a document-based approach to the problem, After this, in section four we will describe the proposed model-based solution. The next section will compare both approaches. Finally, in the final section we will outline the conclusions we have reached and some future lines of work.

2 Context

An electronic invoice is a transaction document that contains billing information in an electronic format. They are the evolution of traditional paper invoices, and as such, they must comply with the same legal restrictions and characteristics. They provide some advantages over conventional invoices, being the more noticeable a reduction in operation costs from not having to deal with big volumes of paper, and a reduction in the time involving the cashing process.

However, there are hundreds of specifications for electronic invoices [1]. This proves to be a major drawback and difficult the interoperability between different countries. Traditionally, one of the most important standards has been the United Nations/Electronic Data Interchange For Administration, Commerce and Transport (UN/EDIFACT) [2]. The aim of this specification is to provide an invoice format suitable for machine to machine (M2M) communications. This format considers multi-country and multi-industry exchange. It was adopted by the International Organization for Standardization (ISO) as the standard ISO 9735.

However, the pass of time led to the massive support of XML-based formats for the industry, thanks to its interoperability capabilities for B2B. Therefore, UN/EDIFACT has been losing traction in favor of newer specifications. Following this trend, and taking into account that UN/EDIFACT is still widely used in Europe an XML version called XML/EDIFACT [3] was released as ISO TS 20625.

Another evolution of UN/EDIFACT is ebXML [4]. ebXML has been designed by UN/EDIFACT's creators and OASIS. Although ebXML was designed to solve the interoperability problems, the lack of implementations and enterprise support have ruled it to the background.

After collaborating in designing ebXML, OASIS worked on their own electronic invoice format, developing Universal Business Language (UBL [5]). UBL is based upon the core components of ebXML but tries to solve its interoperability problems[6]. Unlike ebXML, UBL is greatly supported by many northern European countries such as Norway, Denmark or Sweden, which formed an alliance to adapt UBL to their necessities creating NESUBL[7].

But even with the recent success of UBL, the problem of interoperability between countries still exists, especially in Europe. For example, Spain uses the XML-based

standard Facturae [8] as its main electronic invoice format. Facturae is a format developed entirely by request of the government of Spain, and it is used exclusively in Spain. Moreover, this standard experiences continuous evolution, with three different versions being released (v3.0, v3.1 and v3.2) in the 2007-2009 period. But this is just an example, since Spain is the main country of interest for our research. Some other European countries use their own invoice formats, for instance, Croatia uses its own implementation of ebXML. Although there have been some initiatives to define a common European XML invoice format, none of them have succeeded. At the time of writing of this paper, if a Spanish company, for instance, wishes to issue an invoice to a Danish company, its invoices could be invalid in Denmark as they are not in UBL format. In addition to this problem, Enterprise Resource Planning (ERP) applications are often tied to a single format, which means that companies can't even work with other formats than the one their ERP uses.

There are several research efforts dedicated to solve this problem. CENBII [9] is an initiative to provide public interfaces for interoperability between UN/EDIFACT and UBL. Pan-European Public Procurement Online (PEPPOL [10]) tries to provide an interoperability environment so any company within EU borders can operate with other. The Interoperable Delivery of European eGovernment Services to public Administrations, Businesses and Citizens (IDABC) [11] also tries to establish a common platform for electronic invoices across EU.

The problem of supporting multiple invoice formats is not only a matter of documents which, but also mandates more complex operations, such as watch for the accounting information correctness of the documents (taking into account aspects such as local sums, types of taxes or discounts). The degree of support of those factors widely differs among the different standards, as in many cases they are also related to the national regulations.

Moreover, the security constraints of a financial further complicate invoice management systems. To provide the adequate validation of an invoice content and issuer, electronic invoices are digitally signed for assuring content integrity. The problem with this process is that after signing the invoice, its contents and format cannot be changed. Hence, to enable the conversion between invoice formats the conversion has to be done before the signing and sealing of the invoice.

As we can see, there is a lot of interest in determining a full interoperability environment. Eventually, common grounds will be reached and a unique standard will be used. In the meantime, we are in a transition period in which interoperability is not achieved.

3 Document-based invoice management system

In order to provide a clearer view on the management of multiple formats of electronic invoices, we will present how traditional approaches address the problem.. We use the term document-based approach for these alternatives, as the central elements are the XML invoice documents.

Electronic Invoices document the transactions between a seller company and a buyer company. An invoice management system must allow the seller to issue the

invoice and the buyer to receive and accept it. In addition to supporting this workflow and storing the information, these systems must also perform additional operations over the process, such as checking the invoice document respects the correct format, verifying the authentication credentials of the involved parties, and performing additional validation functions, such as calculating partial and total aggregates, evaluating deduction amounts, or verifying addresses are correct.

In the financial world, it is usually required for the issuer of an invoice to store either the original invoice or information which would allow us to regenerate the invoice. That information is also called the invoice matrix. The former is the easier option to implement. In this case, the invoice is an XML document. Therefore, the target of this architecture is the storage of an XML representing an invoice in a database. However, since support of several invoice formats is needed, the architecture should be able to convert between different invoice formats, previously to storing the invoices.

Invoice transformation is a necessity in many scenarios, as it will be required whenever the two companies involved in the transaction work with different invoice formats. This way, the seller would either have to be able to issue an invoice using the customer's desired format or manage to convert their invoices to that format. This way, we start with an invoice in one format and need a different format to be produced and stored.

The following figure shows how to support the second option internally in the system. Taking as input the XML document with the issued invoice, the invoice is validated, transformed to the desired format, and then stored in the database. During the whole process we are constantly working with XML documents.

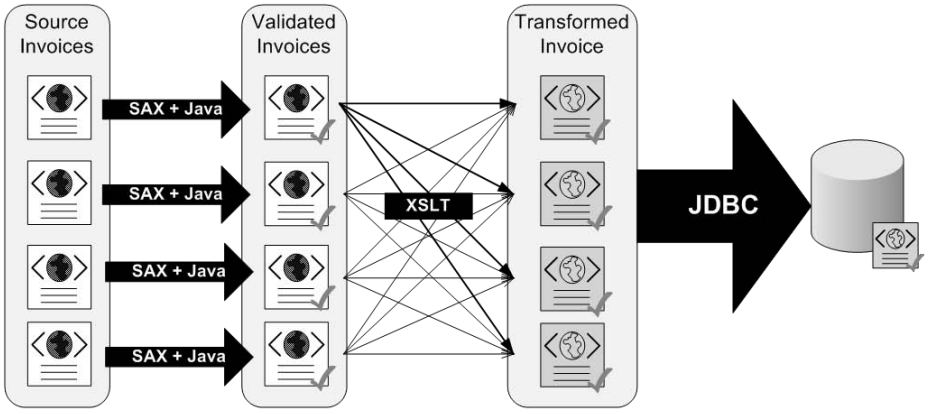

Fig. 1. Document-centric approach.

In this process, the first step is to validate the input invoice. As it has been previously mentioned, there are two types of validation which must be applied. The first one is a syntactic validation, in which we check that the document complies with the format dictated by the XSD schema. Thanks to the availability of these documents, and the existence of mature Java XML libraries[12] this step is not excessively difficult. On top of that, it can be easily applied to every supported

invoice format, as it only needs the schema specification.. However, the second validation step, the accounting validation, presents more problems. Accounting validation consists of checking several amounts, prices and quantities in the invoice in order to rule out any financial problem. This step is extremely important, as once the invoice is transformed to other format, it would be very difficult for the issuer's ERP to change data on an invoice in a format it does not know. On top of that, this process is more complex than simply checking the invoice syntax. To perform this validation, we need to have a deep knowledge of the format we are working with, since we need to know exactly what fields we must check, and how. The relevant fields of the invoice must be extracted using XML parsing libraries such as SAX or DOM, and then some operations need to be carried out in the logic services to check the correctness of the invoice (checking correct application of tax rates, totals, etc…). Clearly, this must be handled separately for each supported invoice format.

After the invoice has been validated, we can proceed to transform it to the invoice format desired by the customer. As current invoice standards are based on XML, we can use XSLT [13] for this operation. XSLT is a W3C standard for transforming XML documents. It uses a template which contains rules written using XPath syntax and is able transform the original document into a new one. Hence, we need to create an XSLT template to implement the transformation between every two invoice formats. Again, this step requires a previous study of both the issuer's format and the customer's format, in order to be able to map fields from one to the other. These templates are not bidirectional, so another template is needed to reverse the transformation. In sum, we need one template per pair of formats per direction. This means that adding new formats to the mix involves creating several templates, whose numbers grow with the number of supported formats exponentially. Since we used a validated invoice as input and can trust the correctness of the transformation template, we end with a validated invoice in the desired format.

After the transformation has been completed, the issuer must sign the invoice before storing or sending it. These operations could not have been carried out before, as even although the security information could be transferred to the transformed invoice, the transferred invoice is a new file and the digital sign would not match the contents of this new invoice.

Finally, the system must persist the generated invoice in a database. There are several ways of performing this operation. The first one is to store the whole invoice as a large binary object (BLOB) in the database. The problem with this solution is that searches in the invoices cannot be performed, since the only way of distinguishing one from another is to get all of them from the database and read its contents. Another approach that eases searches consists in designing the database to mimic the data schema of the format we want to store. That way, every data in the invoice can be searched. The problem with this method is that making that design involves a lot of work which is not reusable. Even more, only one format could be stored this way. A third solution is a mixture of the previous ones. SAX could be used to extract a few selected fields that enable some basic searches, common to all invoice formats. This extracted common data can be then stored in the database linked to the invoice as a BLOB.

4 Model-centric approach

After presenting the current state of electronic invoice formats and briefly outlining the base functions which should be supported by an invoice management system, we will describe in this section our proposed approach. In order to do so, we will first introduce the Model Driven Engineering technologies which can be adopted in this specific context. After that, we will describe our proposed solution.

4.1 Proposed MDE Technologies

We have already mentioned the challenges which must be addressed by invoice management systems.. At this point we will present. the most interesting technologies from the MDE[14] domain with respect to its applicability in this context. It must be noted that our aim is not to apply a complete MDA process to the development of invoice management systems (i.e. generate the complete system from a model, define the PIM and PSM, etc.), but instead to apply MDE techniques for the key domain functions (invoice generation, transformation, validation and persistence).

,In the context we explained that most of invoice formats are defined in XML. Every XML document must follow the rules of an XML Schema Definition (XSD). Therefore, for an XML document to comply with an invoice format, it must follow the rules of that format XSDs. Considering this, to be able to use the definition of the invoice format as provided (an XSD) we need a modeling language and a set of tools that support the transformation from and to that language and an XSD.

From all the tools that provide this capability we have selected Eclipse EMF [15]. EMF is a modeling framework and a set of tools based on the Ecore language. Ecore is a subset of MOF (Meta Object Facility), the metametamodeling language standardized by the OMG to model UML, among others. EMF provides us a mature set of tools built over a metamodeling language with a high degree of expressivity, and with a large and participative community. Among the base features provided by EMF it is of special interest to us is the capability to seamlessly transform between an Ecore model, Java code and XML. This allows us to seamlessly obtain model instances from the received documents and automatically perform the syntactic validation to the handled elements.

Clearly, a mandatory characteristic of an electronic invoice is its validity. It is not only needed for an invoice document to be syntactically correct against its XSD. Other validations need also to be performed over an invoice to be sure that it is correct. For instance, we must ensure that all the lines of products in the invoice, when added, produce a result which is equal to the invoice total, taking into account taxes. To perform validations like this over a model we need a language that enables us to define rules that a given model has to comply with. For this task we have selected OCL [16] (Object Constraint Language). Since OCL can be used with any language written in MOF, any model created with EMF is eligible for its use. Another option would have been Schematron [17], but it is more aimed to work with XML documents. Although the invoices are easily convertible between EMF and XML, it is

preferable to use the models directly. This way, we support the validation of invoices completely at model level.

For the development of the conversion between the different invoice formats we need a transformation language. To this end we have selected ATL (ATLAS Transformation Language). ATL [18] is an open source transformation language developed by the Eclipse Project. ATL designed for extensibilityhas a modular model transformation virtual machine and is therefore very easy to extend. The reason for not using QVT (Query- View – Transformation) [19] instead of ATL is that the former is centered in the Platform Independent Model to Platform Specific Model transformation of MDA. ATL is more suitable to general purpose transformations like the conversion of invoices from one format to another.

One of the most important non-functional requirements of enterprise applications is the persistence of the information. To ease the process of storage of invoices and other financial information using models we decided to use Teneo [20]. Teneo is a database persistency solution for EMF. Starting from an Ecore model it uses Hibernate to generate the relational structure and provide an easy interface for the management of the persisted information.

4.2 Proposed approach

Our proposed approach consists in applying MDE techniques to support the validation, transformation and persistence functions required by multi-format invoice management systems. In order to do so, we will translate the original invoice provided by the issuer's ERP to a model using EMF, and we will work with said model during the process, instead of dragging the original invoice document all over the operation. The next figure shows how the previously described process can be executed using ae model-centric approach.

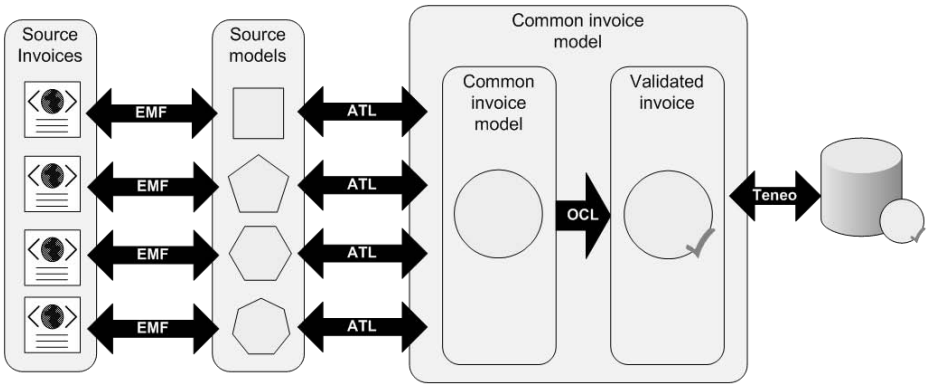

Fig. 2. Model-centric approach.

The first step is to obtain a model instance from the initial invoice document. This is supported with the EMF functionality that, starting from an XSD schema automatically generates a corresponding Ecore metamodel. In addition to that, it can

also generate a marshalling / unmarshalling layer XML documents to/from ecore instances. Therefore, the process of converting the original invoice into a model is performed with little effort.

In this approach, taking into account that we are working at a higher level of abstraction, we have opted for a more extensible approach by defining a common invoice model, This common invoice model would be composed by elements common to all invoices, and all the data that, although not common, is still relevant. This way, every invoice specific model will be transformed to the common format as soon as it is imported into the platform. The common model will be created using EMF from scratch.

The definition of this model is the most time consuming step of the development of this approach. A complex study has to be performed including all the formats that want to be supported by the system. Common concepts have to be extracted from all of them, and a special care has to be put in filtering the relevant fields and reaching a satisfying definition. The transformation from and to this common model must not incur in any loss of information.

With this common model created, we can transform using ATL from every format into it and then perform any other operation with this common model. This greatly eases the adoption of new formats, as only the model for that format and the transformation from and to the common model need to be supplied.

Once we have the common invoice model object for a given invoice, validation can be performed. Syntactic validation is no necessary since EMF automatically checks the syntax when marshalling an invoice. Therefore, only accounting validation is required. This validation is going to be always performed over the common invoice model with OCL. With OCL this validation is performed directly over the model information, using rules that can be easily expanded to include more complex or other types of validations.

Concerning the storage of the invoice data, in the previous approach we stored the invoice, but in this case what we store is the matrix that allows its creation; the common model. For this step we decided to use Teneo, since it offers the opportunity to persist EMF models automatically. This means that all the information in the matrix can be stored in the database in properly structured tables. This way, the information we can search for in the database is widely expanded, and we can produce invoices in any of the supported formats from the data stored.

To do this we will perform the same steps as before, but in reverse order. This is indicated in the figure with the two-headed arrows. The steps will be as follow: 1) The matrix for a given invoice (common model) is obtained from the databases. 2) The matrix is transformed into the desired target format using ATL. 3) The invoice, now in the desired format in EMF, is marshaled into XML. 4) Finally, we perform any remaining step like signing the invoice or sending it to the customer.

5 Comparative analysis

After introducing the two approaches, we will perform a comparative analysis between them. This way, the strong and weak points of each approach can be detected and an informed decision about which one should be implemented can be made.

For this analysis we are going to center in the following aspects: 1) Initial Effort for implementing the approach. 2) Incremental effort for adding another format to an already developed system. For the purpose of this analysis we are going to impose the requirement for a format to be added to the system to be XML-based.

5.1 Initial implementation

Document-centric approach: The reusability for this approach is very low. The effort of developing an initial system with support for several formats is almost the same as the required for adding a new format as many times as format supported. The only reusable bits are the basic underlying infrastructure (persistence driver, external interfaces, GUIs) and the syntactic validation module. This way, the initial effort of this approach can be equated to the number of formats supported.

Model-centric approach: One of the first concerns that need to be taken into account if a model-centric approach wants to be implemented, is the learning curve inherent to the MDE technologies. If the developers are not familiar with them, this initial overhead can be very high. And, although this experience can be reused in the future, it imposes an initial effort that should not be ignored.

But even when the developers are already familiar with these technologies, the initial development of this approach is much more costly than the document-centric one. The main effort is going to be the common invoice format. This development is difficult, requires a good analysis of all the invoice formats, and has to be done by an expert in the field of electronic commerce. In addition to this, OCL rules for validating this model (accounting validation) must be defined and a database table structure has to be created. Luckily, some of these tasks are greatly simplified thanks to tools like Teneo.

5.2 Addition of a new format

Document-centric approach: In this approach the addition of a new format imposes the following developments: 1) Syntactic validation of the XML document. 2) Accounting validation of the XML document. 3) Transformation to and from every other supported format. 4) Persistence of the format.

In this case, the implementation of the syntactic validation is straightforward, since the code existing for any other format can be reused almost completely. However, the accounting validation presents a bigger problem. A profound knowledge of the new format is required and this validation is costly to implement since it will be very different to all the other formats validation.

Transformation presents a similar problem. Knowledge of the new format is required. On top of that, two new transformations (one to and one from) must be created for each other format already supported. This imposes even another requirement: knowledge of not only the new format, but also every other format already supported. This is the dominant cost factor with this approach.

Finally, concerning persistence of the new invoice format, this heavily depends in which of the three solutions proposed have been adopted. If the invoice is stored as a large binary object, the cost is negligible. However, if we need to extract some information or, even worse, create a new table structure, the development effort could be very high.

Model-centric approach:

For this approach the addition of a new format imposes only these two tasks: 1) Generation of the EMF model for the new format 2) Transformation to and from common invoice format.

From these two developments, the first one is the easiest. This can be done almost automatically using the tools provided with EMF. Therefore, the effort is almost zero. Consequently, the only real developments are the transformations to and from the common invoice format. As in the previous approach, this requires knowledge of the new format. But, unlike before, only knowledge of the common format (and not all the other supported formats) is needed. On top of that, only two transformations need to be created.

6 Conclusions

In this article we have presented the main challenges behind electronic invoice management systems, and have presented a potential application of MDE techniques and technologies for addressing them. This way, the management system architecture is built on top of a set of functional elements which handle models of the electronic invoices. At this abstraction level, generation, transformation, validation and persistence are easily supported. This presents significant advantages when compared to an approach which directly manages the base XML documents, thanks to the increased expressivity an extensibility of the model layer.

Although this article is focused in an specific domain (electronic invoice management), we believe the benefits of implementing the domain model and core functional blocks of a system through the use of MDE technologies have greater implications. At the current stage, languages and tools have reached a level of maturity where these specific decisions can greatly improve overall productivity and increase the overall quality of the resulting systems (in aspects such as extensibility or maintainability).

Regarding future work, we intend to expand our model coverage so they don't only capture the electronic invoices but also cover the identity information of the participating companies. This way, digital signature functions could be integrated into the model workflow, and automated validation rules could be defined for the information from each company.

Acknowledgements

The work presented here has been developed in the context of the project Factur@. Factur@ is an IT innovation project funded by Telvent and Comunidad de Madrid. The authors are grateful to Telvent for supporting this work..

References

1. Vanjak, Z.; Mornar, V.; Magdalenic, I.: Deployment of e-Invoice in Croatia. In: Cordeiro, J., Shishkov, B., Ranchordas, A., Helfert, M. (eds.) ICSOFT 2008 - Proceedings of the Third International Conference on Software and Data Technologies, vol. ISDM/ABF, pp. 348—354, INSTICC Press (2008)
2. Berge, J. : The EDIFACT Standards. Blackwell Publishers, Inc. (1994)
3. ISO/TS 20625:2002, Electronic data interchange for administration, commerce and transport (EDIFACT) - Rules for generation of XML scheme files (XSD) on the basis of EDI(FACT) implementation guidelines
4. OASIS's ebXML specification, http://www.ebxml.org
5. OASIS's UBL comittee, http://www.oasis-open.org/comittees/ubl/
6. Tolle, K.: UBL: The DNA of next generation e-Business. In: Information Technology and Control, vol. 37, No. 1, pp. 38—42, Kaunas, Technologija (2008)
7. Northern European Subset, NES; specification: http://www.nesubl.eu
8. Government of Spain's Facturae information: http://www.facturae.es
9. CENBII specification: http://spec.cenbii.eu
10. Pan-European Public Procurement Online, http://www.peppol.eu
11. IDABC commission information: http://ec.europa.eu/idabc/en/home
12. McLaughlin, B., Edelson, J.: Java and XML. O'Reilly Media; 3rd Edition (2006)
13. XSL Transformation specification: http://www.w3.org/TR/xslt
14. Kent, S.: Model Driven Engineering. Integrated Formal Methods. In: Lecture Notes in Computer Science, volume 2335/2002, pp. 286-298, Springer Berlin / Heidelberg (2002)
15. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2008)
16. Richters, M., Gogolla, M.: OCL: Syntax, Semantics, and Tools. In: Object Modeling with the OCL, vol. 2263/2002, pp. 447-450, Springer Berlin / Heidelberg (2002)
17. Dodds, L.:. Schematron: Validating XML Using XSLT. In: XSLT-UK Conference, Keble College, Oxford, England (2001)
18. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: a QVT-like transformation language. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 719-720, ACM, New York, USA (2006)
19. OMG: MOF 2.0 Query / Views / Transformations RFP, OMG Document ad/2002-04-10. (2002)
20. Eclipse Foundation. Teneo: http://www.eclipse.org/modeling/emft/?project=teneo.