Technological Spaces: an Initial Appraisal♣

Ivan Kurtev [1], Jean Bézivin [2], Mehmet Aksit [1]

[1] Software Engineering Group (TRESE), University of Twente, The Netherlands
{kurtev, aksit}@cs.utwente.nl

[2] Faculty of Sciences, University of Nantes, France
bezivin@sciences.univ-nantes.fr

Abstract. In this paper, we propose a high level view of technological spaces (TS) and relations among these spaces. A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference regular meetings. Although it is difficult to give a precise definition, some TSs can be easily identified, e.g. the XML TS, the DBMS TS, the abstract syntax TS, the meta-model (OMG/MDA) TS, etc. The purpose of our work is not to define an abstract theory of technological spaces, but to figure out how to work more efficiently by using the best possibilities of each technology. To do so, we need a basic understanding of the similarities and differences between various TSs, and also of the possible operational bridges that will allow transferring the results obtained in one TS to other TS. We hope that the presented industrial vision may help us putting forward the idea that there could be more cooperation than competition among alternative technologies. Furthermore, as the spectrum of such available technologies is rapidly broadening, the necessity to offer clear guidelines when choosing practical solutions to engineering problems is becoming a must, not only for teachers but for project leaders as well.

1 Introduction

Nowadays software engineers generally have to cope with a large variety of possible solutions when asked to solve a specific problem. Very often the same problem may be solved with the help of different technologies, at different prices, with different qualities. How should we give the advice of using particular technology to solve a given problem? What are the different trade-offs of using one technology instead of another one? If we choose one set of solutions, how far are we captive of this choice when we'll need later to evolve or maintain our solution? These are some of the practical questions one may ask when developing a software solution. In order to provide guidelines that will be more than wishful thinking, we need to start talking explicitly about the characteristics and properties of each technological context and to show how we may provide guidance about using a specific context when solving a particular problem.

We employ the concept of Technological Space (TS) as the central concept in our analysis and comparison. A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. It is often associated to a given user community with shared know-how, educational support, common literature and even workshop and conference meetings. It is at the same time a zone of established expertise and ongoing research and a repository for abstract and concrete resources. Instead of asking the a-priori question of the possible existence of a unifying theory for all technological spaces, we rather ask how to know the advantages and drawbacks of each one and how to suggest practical utilization strategies. It is possible that some conclusions of this investigation will make us aware of some conceptual similarity between the technological spaces.

To give a flavor of the undertaken work, let us initially consider five technological spaces: Programming languages concrete and abstract syntax, Ontology engineering, XML-based languages, Data Base Management Systems (DBMS), Model-Driven Architecture (MDA) as defined by the OMG. Figure 1 provides a global view of these five TSs. It shows that each space is defined according to a couple of basic concepts: Program/Grammar for the Syntax TS, Document/Schema for the XML TS, Model/Meta-Model for the MDA TS, Ontology/Top-Level Ontology for the Ontology engineering TS and Data/Schema for the DBMS TS. Another idea, illustrated by Figure 1, is that no TS is an island. There are bridges between various spaces and these bridges also have particular properties. Some may be bi-directional and some may be one-way bridges. Some operations may be performed easier in one space and the result may then be imported into other space. For such an operation, software engineers need to compare the facility of

♣ Due to space limitations, many examples and suggestions could not be included in this paper. An expanded version of this work may be found at http://wwwhome.cs.utwente.nl/~kurtev/TechnologicalSpaces.doc

achieving it in one space compared to the other one and also to evaluate the export/import facilities between the spaces. In Figure 1, we do not represent all the bridges among the TSs and the figure does not suggest any superiority for any one of them.

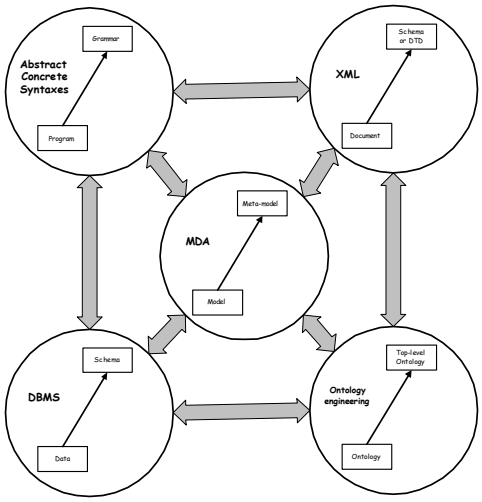

Figure 1 Five TSs and some bridges between them

This paper is organized as follows. After the introduction, section 2 provides a brief insight on some TSs for global understanding. Space limitation does not permit a complete coverage of each technology, but some pointers are provided for further reading. Section 3 presents a comparison of the features of various TSs. Section 4 deals with interoperability bridges among different spaces. Section 5 describes some related work. Finally, section 6 concludes the paper by providing some suggestions on how to interpret this work and how to integrate these preliminary findings in concrete engineering practice.

2 Description of some TSs

Before starting our discussion, we provide a short presentation of some of the TSs that will be referred to in the rest of the paper.

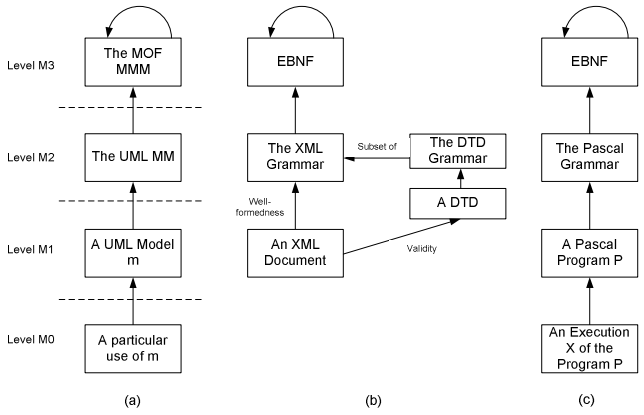

Figure 2 MDA, XML and AS Technological Spaces

2.1 The MDA TS

Model Driven Architecture (MDA) [22] is an approach recently proposed by OMG[1]. According to MDA, the software development process is populated with a number of different models, each representing a particular view on the system being built. Models are written in the language of their meta-model. There are several standard meta-models available in the context of MDA, the most popular is the UML meta-

[1] An alternative approach for manipulation of models has been proposed in [3].

model [18]. This organization is accommodated in a framework sometimes called the MDA four-level modeling stack (Figure 2a). The lowest part called M0 corresponds to the real world. All models are at the level M1 and the definition of languages (meta-models) used to create these models are at the level M2. Finally there is also a special language used to define the meta-models. It is called the MOF (Meta-Object Facility) [17] and is qualified as a meta-meta-model. The MOF is self defined and constitutes the level M3.
One particular objective of the MDA is to separate models that are neutral descriptions of the system (PIM or Platform Independent Models), from the models that are bound to a particular technology (PSM or Platform Specific Models). The objective is to define standard processes to obtain PSMs from PIMs, hopefully by some kind of automatic generation.

2.2 The XML TS

Extensible Markup Language (XML) [25] is a framework for definition of markup languages standardized by W3C. It is widely accepted as a standard for representation and exchange of structured and semi-structured data. XML document is the central concept in the XML TS. Documents are written in a syntax constrained by well-formedness and validity constraints. Well-formedness constraints are defined by the XML grammar rules, whereas the validity constraints are defined in a separate document called document schema, which is written in a given schema language (Document Type Definition (DTD), XML Schema language, etc.). Figure 2b illustrates the relations between an XML document, the XML grammar and a document schema defined as a DTD.

2.3 The Abstract Syntax TS

AS TS is the space with the longest history rooted at the days when computer engineers started building abstractions over the native machine language and expressing these abstractions with languages closer to the problem domain.
It is based on solid theoretical foundations: context-free grammars for specification of language syntax; a number of formalisms for specification of language semantics (attribute grammars, denotational and action semantics, etc.). A number of programming paradigms can be observed: imperative, declarative, functional, logic, procedural, object-oriented programming.
The AS TS is organized around the concepts of program written in a given programming language whose syntax is formally specified in a grammar. What characterizes this space is that the goal is to deal with executable programs. These concepts and the relationships between them are shown in Figure 2c by using the programming language Pascal as example.

2.4 The Ontology TS

According to Gruber, ontology is an explicit specification of an abstraction. The dimension of consensual agreement (usually normative) should be added to this definition.
The field of ontology engineering may be considered as a subfield of knowledge engineering. The two main dimensions are representation and reasoning. According to the importance given to each, one may find several approaches like terminological logics, conceptual graphs, etc [23]. Several languages are used to express ontologies like KIF (Knowledge Interchange Format). A short introduction to this field may be found in [9].

As illustrated by the examples, the TSs correspond to the main area of activity and expertise of computer engineers. A generalist engineer has some exposure to the majority of these fields, while many specialists are more knowledgeable in only some of them. Obviously, we do not aim in this paper to give a complete list of all the TSs. Nevertheless, we think that these example spaces provide sufficient insight for the purpose of this paper.

3 Comparing Spaces

Table 1 summarizes how some technologies do better on some problems than others. This table is given only for the illustration of the proposed approach. Because of the space limitations we will discuss only a part of the issues of Table 1.

Table 1 A comparison between some TSs

	XML	MDA	AS	Ontologies
Executability	Poor	Poor	Excellent	Poor
Aspects	Good	Excellent	Poor	Fair
Formalization	Poor	Poor	Excellent	Fair
Specialization	Fair	Good	Poor	Fair

Modularity	Good	Good	Good[2]	Poor
Traceability	Good	Fair	Poor	Excellent
Transformability	Excellent	Fair	Fair	Fair

- *Executability*: Among the spaces being described, only AS TS has precise executable semantics. In contrast, the XML TS is a framework for definition of standardized syntax only. The need for some form of executability can be identified in the other spaces as well. DBMS TS uses stored procedures mechanism; Action Semantics for UML is intended to add executable actions and procedures to the models; XML DOM trees can be scripted to add some behaviour as a response to user events.
- *Modularity*: The story of computer science shows that much energy has been put on the idea of modularity in programming languages since the 60's. The work of Parnas, Wirth and many others has progressively elaborated on the *def/ref* or *import/export* semantics of the early assembly languages and later on the module feature found in Modula, Oberon, etc. This has given birth in turn to the package notion of ADA and Java that has been borrowed in turn by modeling languages like UML. The same notion is called namespace in C# and the concept of namespace is also central to the XML technology.
- *Content/Presentation Separation*: It seems that the XML community was the first to encounter the problem of separation of content from presentation. In the HTML language, both parts are mixed. The design of XML is focused on the description of content while other languages (CSS, XSL FO, etc.) are specialized on presentation.
  In MDA the same problem was encountered. When exchanging UML models between tools, the logic of diagrams was the essential information but had to be completed by much presentation information (how box and lines are drawn?, etc.). In UML 2.0, it is possible to separate the logic of a model from its presentation. Intentional Programming (IP) [21] has among its goals the separation between the abstract syntax and the concrete notation allowing multiple representations of the same intention (textual, visual, etc.).
- *Traceability*: The idea of traceability is a very important concept. However, it is a particular case of the specification of correspondence between various contexts. It seems to us that the basic relation of co-reference in conceptual graphs provides one of the most general expressions of this.
  The notion of traceability is very important in MDA since moving from a model to another one is usually associated to a design decision or to a transformation. The source of each element should be precisely identified and then the process could be reversed. Maintaining traceability links between elements of all models is an important engineering challenge.

We have seen in this section that very often, when we consider a problem inside one particular technology, it corresponds to similar problems within other technologies. The way to solve this problem is not always the same.
Previous discussion suggests that technologies can learn from each other and one may be improved by inspecting how other technologies are proceeding to solve certain problems.

4 Bridging Spaces

One thing is to discuss the various possibilities of TSs and another is to handle the real interoperability problems between them. In this section we'll deal with the second part, i.e. how we can import or export results obtained in one technology from or to another different technology. A full coverage of the bridging between spaces would include descriptions of the available techniques and tools, encountered problems and examples. Because of the lack of space we will limit ourselves to giving few examples when moving to another space to solve a given problem is the essential part of the solution.

4.1 Examples of Bridging between Spaces

In Business-to-Business eCommerce integration problems caused by the difference in XML syntaxes of the exchanged documents are typically solved with XSLT transformations. These transformations tend to be complex because several tasks are mixed in the XSLT rules: terminology translation, syntactical translation between the XML syntaxes, etc. One proposed solution reduces the complexity by splitting these tasks into different layers [14][16]. The document schemas (e.g. DTDs) are abstracted to a conceptual representation (e.g. RDF Schema). Then, the transformation between the two conceptual representations/ontologies is

[2] Note however that if modularity features have been allowed at the level of programs, there has been little effort in providing them at the level of grammars, for handling specialization or aggregation for example.

specified. XSLT transformation can be automatically obtained later. Thus, the translation is specified in a new TS (Ontology engineering).
Another illustration of the potential of bridging is migration between two programming languages. Recently an approach to migrate Java applications to C# applications was proposed as a set of tools by Microsoft. The corresponding JUMP framework (Java User Migration Path) is entirely situated in the programming language (Abstract Syntax) TS. Alternatively, it is also possible to perform a reverse engineering operation from Java to a UML-like model (R operation), followed by a forward engineering operation to generate the C# program (F operation). The direct operation will be different from the combined R+F operation.
Finally, it is worth to mention the application of bridging in the design of various software artifacts where the design starts in one space and the desired result is obtained by transformation (eventually automated) to another space. A typical example is the design or relational schemas from ER models.

4.2 Discussion

Technological spaces can be classified according to their level of abstraction [19]. If more semantics can be expressed in a given TS then it is said to be at higher level of abstraction. MDA and Ontology engineering TSs can be regarded as high-level (abstract) spaces while the XML, RDBMS and AS TSs can be regarded as low level (concrete) spaces. It seems that this distinction is manifested in the most of the characteristics of bridging. Operations in a concrete space sometimes involve semantic aspects, which are not explicitly represented. Moving to a suitable abstract space to deal with the semantic issues can reduce the complexity. Also, software engineers must be aware of a possible loss of semantics when importing from an abstract to a concrete space. In the opposite direction, we observe difficulties to induce the semantics only on the base of the source artifact.
Another important aspect of bridging is the presence of multiple alternative transformations from one space to another. They can be compared on the base of certain quality factors such as adaptability, extensibility, compactness, etc. However, in the current transformation techniques, the notion of the quality of the result is more implicit than explicit. Designers make selection intuitively rather than following a methodology. The exploration and evaluation of the set of alternatives is a complex task and can be an obstacle for finding a suitable transformation. This problem is identified in several concrete areas such as design of XML schemas from models [4][8] and deriving object-oriented designs from abstract specifications[24].

5 Related Work

The focus in this paper is on providing a higher-level comparative view over a number of technologies and the potential for interoperability between them. We do not provide many details about concrete technologies and bridging. This is a subject of separate research activities and we refer the reader to some ongoing research on integrating different technologies. Semantic Web [2] initiative aims at enhancing the WWW with semantic information by integrating the achievements from both XML and Ontology engineering fields. Integration between UML and Ontologies is studied in [5][6][7][11][15]. The problem of storing XML documents raised an interest from the Database community [10][20]. Possible mutual improvements between Databases and Ontologies are discussed in [12][13]. An interesting report on the interest of using the Ontology TS to help with the integration of XML web resources may be found in [1].

6 Conclusion

This work is an initial investigation in the field of technological spaces. In the process of performing this study, we discovered several facts.
By comparing several technological spaces, we see that similar problems are solved in different ways. Abstracting these problems seems to be an interesting and productive approach.
When we consider the various TSs, one idea that arises is the notion of typing/meta level. If an entity is found at one level, the definition of this entity (its type) may be found at the immediately upper level. The levels are called meta-modeling layers in MDA and we found similar organization in other fields, even if it is not always completely explicit (see Figure 2).
It seems that the pattern of research discovery changed in the last decade. Until now, inventions were often made within the strict boundary of a given engineering field (operating systems, compilers, data bases, etc.). If we look at what is happening now, it seems that innovation often starts at the boundary between several different engineering fields. The example of XML is well known for that because it took birth between network application, hypermedia and document processing before becoming itself recognized as a specific zone of research and expertise. The synergy between different fields seems to become the main innovation factor. Unfortunately the research work is still mainly located to historical fields and "transversal" results are not very frequent.

References

[1] Amann, B., Beeri, C. Fundulaki, I., Scholl, M. *Ontology-Based Integration of XML Web Resources* In Proceedings Intl. Semantic Web Conference 2002 (ISWC 2002).

[2] Berners-Lee, Hendler, & Lassila. *The Semantic Web*, Scientific American, May, 2001.

[3] Bernstein, P.A., Levy, A.Y., Pottinger, R.A. A Vision for Management of Complex Models Technical report MSR-TR-2000-53, Available from ftp://ftp.research.microsoft.com/pub/tr/tr-2000-53.pdf

[4] Bird, L., Goodchild, A., Halpin, T. (2000). *Object Role Modeling and XML-Schema*. In Int. Conf. On Conceptual Modeling (ER), Salt Lake City, UT

[5] Cranefield, S. *UML and the Semantic Web*, Proc. of the International Semantic Web Working Symposium, Palo Alto, 2001. Available from http://www.semanticweb.org/SWWS/program/full/paper1.pdf

[6] Cranefield, S., & Purvis, M. *UML as an Ontology Modeling Language*, Proc. of the Workshop on Intelligent Information Integration, 16th Int. Joint Conference on AI (IJCAI-99). Germany, 1999.

[7] Dutra, M., *UML for Knowledge Representation*, OMG's Second Workshop on UML for Enterprise Applications, December 2001

[8] Embley, D., Mok, W.Y. *Developing XML Documents with Guaranteed "Good" Properties.* 20th Int. Conf. on Conceptual Modeling (ER), Yokohama, Japan, 2001

[9] Fensel, D. *Ontologies: A silver Bullet for Knowledge Management and Electronic Commerce*, Springer, August 2000.

[10] Florescu, D., Kossmann, D. *Storing and Querying XML Data Using an RDBMS*. IEEE Data Eng. Bulletin 22**,** 1999

[11] Guarino N., Welty, C. *Towards a methodology for ontology-based model engineering*. In Bézivin, J. and Ernst, J., eds, First International Workshop on Model engineering, Nice, France, (June 13, 2000).

[12] Meersman, R. *Can Ontologies Learn Something from Database Semantics*. Dagstuhl "Semantics for the Web" workshop 20-24 March 2000

[13] Meersman, R. *Ontologies and Databases: More than a Fleeting Resemblance*. Rome OES/SEO Workshop 14-15 September, 2001

[14] Melnik, S., Decker, S. *A Layered Approach to Information Modeling and Interoperability on the Web*, Proceedings of the Workshop on the Semantic Web at the 4th European Conference on Research and Advanced Technology for Digital Libraries, Lisbon, Portugal, September 2000

[15] Melnik,S. *Representing UML in RDF*. Available from http://www-db.stanford.edu/~melnik/rdf/uml/

[16] Omelayenko B. and Fensel D., *A Two-Layered Integration Approach for Product Information in B2B E-commerce*, In: K. Bauknecht, S. -K. Madria, G. Pernul (eds.), Electronic Commerce and Web Technologies, Proceedings of the 2nd Int. Conference on Electronic Commerce and Web Technologies, Germany, 2001

[17] OMG/MOF *Meta Object Facility (MOF) Specification*. OMG Document AD/97-08-14, September 1997.

[18] OMG/UML *Unified Modeling Language Specification*. OMG Document AD/ 01-09-67, September 2001.

[19] Peltier, M. & Ziserman, F. & Bézivin J. *On levels of model transformation*, XML Europe Conference, Paris, France, June 2000

[20] Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., Naughton, J. *Relational Databases for Querying XML Documents: Limitations and Opportunities*. In: VLDB, Edinburgh, Scotland, 1999

[21] Simonyi, C. *Intentational Programming – Innovation in the Legacy Age*. Position paper presented at IFIP WG meeting, June 1996

[22] Soley, R. and the OMG staff *Model-Driven Architecture*. OMG document. November 2000.

[23] Sowa, J.F. *Knowledge Representation Logical, Philosophical, and Computational Foundations*. Broks/Cole, Thomson Learning, 2000.

[24] Tekinerdogan, B. *Synthesis-Based Software Architecture Design*, PhD Thesis, Dept. of Computer Science, University of Twente. 2000.

[25] W3C *Extensible Markup Language (XML) 1.0*, February 1998. Available from http://www.w3.org