HETEROGENEOUS SOFTWARE TOOL SEMANTIC INTEGRATION FOR CONCURRENT ENGINEERING: A MULTI-MODEL APPROACH

Jérôme BOULENGER †, Bernard ESPINASSE † and François B. VERNADAT ‡

† DIAM-IUSPIM, Université Aix-Marseille III, Domaine Universitaire de Saint-Jérôme, F-13397 Marseille Cedex 20, France
‡ LGIPM, Université de Metz / ENIM, Ile du Saulcy, F-57012 Metz Cedex 1, France

KEYWORDS
Heterogeneity, semantic integration, ontology, information sharing, knowledge sharing.

ABSTRACT
Especially in an extended or virtual enterprise context, concurrent engineering has to cope with heterogeneous software tools. To integrate these heterogeneous tools, we propose in this paper a bottom-up and multi-model approach to realize such an integration at a semantic level. First we develop the main limitations of the previous top-down approaches of the tool integration and why a bottom-up approach is necessary. Then we develop our approach which takes into account the conceptual model of each tool to realize a relevant semantic integration and which is based on three levels of ontological models (tool, domain and generic). Next we develop static and dynamic aspects of the semantic integration realized according to this new approach. Before concluding and presenting the perspectives of our research, we present the outlines of a multi-agent architecture suitable to support such a semantic tool integration.

INTRODUCTION
One of the main characteristics of Concurrent Engineering (CE) is the integration of all development activities through integration of the participating teams, throughout the product life cycle (Kusiak 1993). CE is an effort both in terms of organization and technology. As far as organization is concerned, we are mainly interested in interdisciplinary and multi-enterprise projects, where each enterprise technical, organizational, cultural and sociological backgrounds are dissimilar, and the collaboration duration is short-term or medium-term. Our proposal takes place in the virtual and extended enterprise framework. In such case, partnerships are mandatory, but the way enterprises collaborate must not disturb or influence internal technical and tools choices. If CE assumes strong integration of the teams, this integration is not necessarily achieved by preliminary techniques and tools harmonization. Indeed, in case of short-term partnership, harmonization costs are most often unacceptable and enterprise agility decreases. Moreover, changing tools may imply a partial loss of enterprise know-how. Therefore, it is important to preserve day-to-day engineers tools and to integrate them, i.e. allowing CE in an heterogeneous environment. We must integrate heterogeneous tools, coming from various trades and vendors, based on different models and frameworks, and using different access methods.

Technologically speaking, some studies have investigated computer support for CE (Ramana et al. 1993), systems integration in CE frameworks, knowledge and information sharing in an heterogeneous environment (Cutkosky et al. 1993; McGuire et al. 1993).

There exists a strong relationship between information sharing and software tool integration. Until now, integration framework has favored top-down approach. This approach consists in the definition of a common global model, attempting to bridge various disciplines. The main drawbacks are: (1) distinctive tool features are not taken into account, (2) tool heterogeneities are ignored and (3) the exchange level and the quality of information sharing decrease.

We believe that a complementary approach is desirable. Starting from tools and trades, this bottom-up approach aims to express and preserve each tool’s own model, and draw relationships among them, in order to realize information sharing at this semantic level. Such an approach is suitable for different integration and interoperation architectures.

The second paragraph deals with the top-down approach limitations, enumerates tool heterogeneities that such an approach cannot take into account, and concludes with knowledge-based integration potentialities.

The third paragraph develops a bottom-up approach, used to define one model for each tool to integrate, and to establish relationships between models. This approach is based on ontology development and reuse, for tool integration. Other static and dynamic integration aspects are also mentioned and exemplified.

The fourth paragraph drafts an example of multi-agent architecture to show multi-model based software tool integration feasibility, and modalities of this integration.

As a final conclusion, we present the foundations of our model for integration, currently under definition and formalization.

TOP-DOWN APPROACH LIMITATIONS FOR HETEROGENEOUS SOFTWARE TOOL INTEGRATION

Previous works in the field of heterogeneous software tool integration are often based on global, unifying, or federating model definition to allow product information and knowledge sharing (Petrie 1992; Vernadat 1996). This model may come from standards, like IGES or STEP (Bloor and Owen 1991), from integrating infrastructures (Cutkosky and Tenenbaum 1992; AMICE 1993; Bounab et al. 1993),
or from emerging consensus (Olsen et al. 1994); however this model is imposed to software tools. Unfortunately, these frameworks deal only superficially with heterogeneity, and jeopardize effective information exchange and sharing.

**Heterogeneities between software tools**

In a collaborative heterogeneous environment, tools suffer from three main heterogeneities: semantic heterogeneity, purpose heterogeneity and referential heterogeneity (Roda et al. 1993; Grashoff 1995). Each tool has its own conceptual model. Two tools without “conceptual overlapping” cannot exchange information. We distinguish three main semantic heterogeneity reasons: different points of view, different concepts refinement, different structures.

Referential heterogeneity results from (1) non homogeneous naming space of tools, (2) tools identities handling idiosyncrasies and (3) partial overlapping concepts (i.e. semantic heterogeneity). Whereas information sharing is possible only if each tool properly identifies the information it generates or uses, and if coreferentiality links between tool identifiers are maintained throughout the project.

Each tool holds a set of operational primitives for some purposes, performed at user’s or other tools’ request. The difficulty is to find if two tool purposes overlap (respectively complete one another). If they do, it means that some tool operational primitives perform the same action (respectively complementary actions), according to tool conceptualization and referential systems. While our main interest is information and knowledge sharing, purpose heterogeneity relates more to distributed cooperating systems.

**Information and data integration levels**

Previous works on software integration showed that information and data integration could be considered at different levels (Figure 1) where each level contributes to data and information exchange or sharing improvement (Brown and McDermid 1992).

![Figure 1 - Information and data integration levels.](image)

At the support level, tools do not share data but files which are data and information supports. Each tool is responsible for file content interpretation, and they are frequent incompatibilities.

Lexical integration of softwares enables them to share the understanding of files structures and formats. But software are often unaware of the content of some parts of shared file. For example the UNIX software chain for document publishing (refer, tbl, eqn, pic, troff) is a lexical integration where tools operate successively on distinct part of a file for a specific operation (respectively bibliographic references, tables, mathematical formulas, pictures and document up-making).

At the syntactic level, software share the same data structures and related rules: a data model. But more often, the tool models are richer than the common syntactic data model. Then some information may be lost during the exchange: this semantic loss is the main problem in heterogeneous software tool integration.

At the semantic level, integrated software must agree with data model and also with the semantic definition of this model in order to avoid semantic loss.

Those different integration levels and strategies allow different integration frameworks developed hereafter.

**Main frameworks for software tool integration**

These frameworks attempt to define a homogeneous space, called the naming space. But they are inadequate because of tool specificities.

Tool integration was improved by data exchange standards (e.g. SET, IGES) and later, by STEP Application Models. The former follows an horizontal approach from tool to tool, staying at the syntax level. The latter is a top-down approach, starting from users needs and attempting to add more semantic accordingly. However, universal and rigid modalities may discourage teams looking for appropriate integration and adaptability.

Consensus and related projects demonstrate conceptual models usefulness for defining knowledge common to team members, and susceptible of sharing (Olsen et al. 1994). Those conceptual models composed of concepts, relations, functions and axioms, are defined in common ontologies. A common ontology defines sorts of exchanged and shared information agreed by users of a community and, consequently, by tool community (Gruber 1993a).

Ontology provides a different way to integrate tools, affording more declarativity to improve the exchange level. However, as experiments in ontological use for CE showed it, common ontology definition and adoption requires team members to reach a precise consensus. On the one hand, this consensus is tricky to reach, because of various disciplines and organizational cultures involved. On the other hand, this consensus does not fit well over model specific tools. Tool users must always remember the necessary adaptation between tool specific model and common model.

Whatever the way consensus is reached, through standards or common ontologies, it is at the semantic level. But finally, the integration coming from this consensus is implemented at the syntactic level, through pre/post-processors or translation procedures.

In our opinion, defining integrating models like STEP Application Models or common ontologies is not a way to treat tool heterogeneities but just an attempt to reach a consensus between users. On the contrary, software vendors and implementors try to differentiate
one’s software tools from competitors one’s, adding new features, functionalities and enhancing representation capabilities. This way, standards definition seems to be a continuous race against obsolescence, with regard to the constant evolution of software tools. The question is: how to reconcile this two antagonistic trends? Our answer is a multi-model approach for semantic integration, allowing and taking into account software tool peculiarities.

A MULTI-MODEL APPROACH FOR SOFTWARE TOOL INTEGRATION

As mentioned before, integration processes based on exchange standards or common ontologies start from a unique model and postulate that tools only need to exchange common information. Therefore, information is represented in the common model. A complete and unique model integrating viewpoints is necessary to represent and keep a coherent product model (Thomas and Rozenblit 1995). Unfortunately, using such a common model neither allows to (1) correctly tackle referential heterogeneity, (2) nor avoids semantic loss during information sent back and forth between tools.

We believe that software tool integration must be considered from the point of view of each tool. The matter is to define a model for each tool in order to represent and clarify concepts it handles, as well as implemented operational rules. This way, consensus is about finding correspondences between concepts stemming from different models (i.e. one-to-one mapping), instead of consensus on a common model. We may remark that correspondence consensus is a local one, easy to reach, and common model consensus is a global one, more tricky to reach.

A multi-model semantic integration framework

Examples of cooperation in a multi-model multi-ontology environment are given in (Knoblock and Ambite 1997; Farquhar et al. 1995). They consist in defining ontologies, and mapping concepts (i.e. ontology correspondences). When a simple correspondence cannot be found, translating (rewriting) rules are defined.

In the context of a set of software tools to be integrated, the problem is to achieve correspondence between tool models. For the purpose of efficiency (i.e. knowledge reuse), tool models may derive more general models. As a result, general models may be used as pivots to find and express correspondences between tool models.

For incompatibility reasons, correspondences between tool models may be difficult or impossible to find (e.g. CSG and B-Rep geometric models). Therefore, it may be necessary to define inter-model translating rules, called articulatory axioms or articulatory rules in the literature. An articulatory axiom aims to define conversion/adaptation to be applied to model concepts, in order to obtain an information semantically equivalent at best, semantically coherent at least, from one model to another. More often it concerns structural transformations (Farquhar 1995).

We consider a three-stage model for software tool integration (Figure 2) inspired from previous works on artificial intelligence, precisely on knowledge representation (Wielinga and Schreiber 1994). A tool may be defined by a terminology and a model describing tool handled concepts as well as concept organization (e.g. typology, composition links...). Tool model is terminology independent and may be applied to different tools distinguishable only by their own terminology.

Tool model semantic, i.e. the set of defining rules about functioning and information input-output validity conditions, is inherited (and if necessary adapted) from the domain ontology the tool is about. A domain ontology describes a field of knowledge (scientific, technical, geographic...) through concepts, concepts relationships and axioms (rules) definition. An axiom sets validity conditions for information which instantiate ontology concepts.

A domain ontology may be defined from other or from generic ontologies, defining templates or knowledge building-blocks easy to reuse.

Tools use their own data model and data format. The first benefits of using tool models is the ability to represent information generated or handled by the tools in an homogeneous environment, at the data model level.

A tool model is defined from a terminology and a domain ontology. A software tool commits to its model, i.e. (1) the tool model is sufficient to represent important characteristics and properties of the tool, in order to integrate it and (2) observable actions of the tool are consistent with the definitions in the tool model (Gruber 1993a).

Nevertheless, tool models are potentially heterogeneous at the conceptual and semantical level. Tool models are related through equivalencies defining mappings between concepts from different tool models (Figure 3). Concept mappings are established in different ways. If tools commit to the same tool model, the correspondence is implicit (case A). If tools commit to different tool models then the correspondence may be either defined directly between tool models (case B) or inherited from a common domain ontology (case C). In this last case, tool models derive or specialize the same domain ontology, enabling to infer some mappings at tool model stage. At last, if tool models derive or specialize
distinct domain ontologies related through mappings, then those mappings may be derived in order to be used at the tool model stage (case D).

Once this conceptual framework define, we have to study static and dynamic aspects of tool integration.

**Static aspects of tool integration**

In order to integrate software tools, we must define tool models, domain ontologies, mappings between tool models and mappings between domain ontologies. Two tools may communicate only if concepts from the sender tool model can be translated into concepts of the receiver tool model (Dowell et al. 1995). We introduce the notion of semantic path: a **semantic path** is a sequence of relations (derivation, specialization, equivalence) chained to link two tool models. Translating a set of concepts from one tool to another is achieved on applying semantical, structural and terminological adaptations enumerated all-along the semantic path.

Verifications must be undertaken to assure tools communication and translation without incoherence nor weakness:

- tool model (respectively domain ontology) coherency and completeness;
- semantic paths linking concepts from communicating tool models;
- consistency of concepts linked by semantic paths (i.e. no contradiction).

We have to take steps against broken semantic path or semantic path narrowing (i.e. some concepts cannot be translated). For example, create new mappings, add new concepts in existing ontologies, create new transient ontologies... Tool models may be incompatible with each other (e.g. CSG and B-Rep geometric models). One solution mentioned above consists in defining articulatory axioms between incompatible models (respectively ontologies).

**Dynamic aspects of tool integration**

Tool integration aims to allow information exchange and sharing. Insofar, as tools are heterogeneous, information must be translated. Translation takes place all-along a semantic path, through concept instance adaptation, from concepts of the sender to concepts of the receiver. The kind of adaptation is expressed inside the semantic path. During translation and instance adaptation, adapted instances must be checked to verify that neither axioms from receiver tool model, nor axioms from intermediate domain ontologies through which they are transferred (i.e. transient ontologies) are violated. During information adaptation, new instances relating to intermediate or final models are created, identified and stored. Identifiers for originating, intermediate and final models instances are all different. Therefore, identifiers coreferentiality must be handle to allow information sharing (i.e. stating that some instances identifiers used by distinct tools, refer to the same concept representative or object).

The coreferentiality link is used also to mark and store the semantic path. This way, if after a translation and for some semantic reason an information get unavailable for the receiver, then it can be restored when sent back, because the coreferentiality link allows indirectly to recover related information. Coreferentiality handling helps to tackle semantic loss by preserving a link between adapted instances of the same concept representative.

**A simplistic example of semantic integration**

Let us consider two CAD/CAM tools to integrate. Tool A is a drawing tool which manipulates geometric figures like rectangle, polygon, ellipse, line and arc. Tool B is a profile editor and a program generator for CNC, and is only aware of lines and arcs used for metal sheet profiles representation. Model of tools A and B are derived from a common domain ontology for 2D geometric modeling (Figure 4). We are in case C of Figure 3.
In our simplistic example, a triangle is sent from tool A to tool B. In tool A this triangle is represented as a polygon. Following the correct semantic path, the triangle in A representation is translated into a set of lines in B representation (Figure 5). Some coreferentiality links maintain a relation between the different instances obtained by adaptation all along the semantic path. These instances represent the same concept representative, according to their specific model or ontology.

After some modifications of the lines with tool B (e.g. resizing) and once the lines are sent back from B to A, it is easy to recognize the triangle in tool A representation; the coreferentiality link is useful to tackle semantic loss and to recover some initial information ignored by tool B.

Moreover, it is also possible to gain some semantic, with regard to an initial representation. For example, five lines in B representation send to tool A may be adapted as a polygon in A representation, if they conform to the axiom express in the geometric ontology, stating that a closed broken-line is a polygon. Concepts, equivalencies, derivation links and axioms defined in the geometric ontology are useful to discover the real nature of some entities. This relates to instance classification, a reasoning mechanism in artificial intelligence (Haton et al. 1991).

**A MULTI-AGENT INTEGRATION ARCHITECTURE**

Once the integration model issues is settled, we can envisage to demonstrate concretely how suitable the approach is for integrating CAD/CAM tools. As previous experiments showed it, a multi-agent system (MAS) lends itself to tool cooperation (Lander 1997). Some proposals based on MAS put multi-ontology systems forward, to ease agent information search and exchange (Barbuceanu an Fox 1994; Takeda et al. 1994; Knoblock and Ambite 1997). To a certain extent our view is similar, but integrated tools are not developed nor modified especially for cooperating.

The underlying infrastructure copes with tool integration and cooperation.

We envision a multi-agent architecture akin to federation system (e.g. PACT [Cutkosky et al. 1993]), also suitable to integrate non cooperating stand-alone tool (Figure 6). A federation system has no shared facility for data storage. Rather, data and information are handled locally and agents communicate through message passing. In such architecture, collaborative intelligence mainly resides in support structures that are domain-independent, viz. system agents. For example, connectivity and message routing may be handled by system agents called facilitators that can be programmed to (1) implement various routing protocols, (2) forward messages to appropriate agents (assertions, requests, answers…), (3) act as repositories for knowledge about what information and services are available for application agent… In this way, an application agent interact directly with one facilitator, and indirectly with the rest of the community.

To ease software integration in a MAS, we propose to use two other system agent types: mediator and agent for integration and cooperation (AIC). We entrust mediators with the task of translating between tool internal data representation and common data model. Translation is done with respect to tool terminology, rules and identification techniques, defined and gathered in a tool model. Mediators are depository for tool models.

For the purpose of information sharing and once translation is done, one or more AIC are put up to information generated or used by application tool. Basically, an AIC consists of three information types: (1) necessary knowledge for the representation of the project frame and the cooperating tools (i.e. domain ontologies and tool models); (2) information generated or used by software tools and represented in accordance with their specific tool model (i.e. concept instances); (3) information identifiers specific to each tool, and coreferentiality link between identifiers originating from distinct tools but referring to the same information.

---

**Figure 5 - Example of semantic path and concept instance translation/adaptation.**
CONCLUSION AND PERSPECTIVES

Software tool integration is most often based on top-down approaches. In this paper we have shown their limitations. Therefore, we propose a bottom-up and multi-model approach offering several advantages: (1) it is suitable to integrate all tool types, and even those traditionally difficult to integrate (legacy systems, platform-independent tools...); (2) it allows to preserve users tools and, consequently, furthers enterprise knowledge and experience preservation (know-how, tools, previous studies and works...); (3) it really takes tools into account and, consequently, tool users and project members; (4) it enables a fast integration obtained with local consensus and knowledge reuse; (5) the translations obtained are coherent; (6) this approach allows a real information sharing through coreferentiality handling, useful to tackle semantic loss and propagate modifications. Moreover, the scalability and flexibility of our integration approach is suitable for any specific CE project, in contrast with more normative and maximal but comforting practices (e.g. STEP). It is based on defining and using ontologies necessary for integration (Gruber 1993b) and does not exclude ontology reuse, whatever their abstraction and genericity levels may be (Fox and Grüninger 1994).

A product data manager system (PDM) is useful in CE projects. Our semantic integration approach is also suitable to partially compensate the absence of a PDM or to harmoniously complete its use. In fact, PDMs often integrate data at the support level only (Figure 1). On the contrary, our semantic integration approach enables to consider information inside a document. Hence, sharing is achieved in information detail, not only at document level.

We are currently defining an operative model for heterogeneous software tool integration according to the approach developed in this paper. The model definition and formalization is in progress. It is articulated around the three levels of ontological models. Two specific languages are defined and used. The first is a modelisation language for ontology definition inspired by Ontolingua (Gruber 1993b). The second allows to express articulatory axioms between concepts from distinct ontologies (Farquhar et al. 1995).

The development of the framework is based on real case in the field of computer aided design, computer aided manufacturing and production management.

REFERENCES


