Requirements Modelling and Formal Analysis using Graph Operations

B. KAMSU-FOGUEM[*]† and V. CHAPURLAT‡

†Laboratoire Génie de Production - Ecole Nationale d'Ingénieurs de Tarbes
47, avenue d'Azereix - BP 1629, 65016 Tarbes Cedex France
Tel : (33) 6 24 30 23 37 Fax : (33) 5 62 44 27 08

‡LGI2P - Site EERIE de l'Ecole des Mines d'Alès, Parc scientifique George Besse,
30035 Nîmes cedex 1 France

Abstract

The increasing complexity of enterprise systems requires a more advanced analysis about the representation of services expected than is currently possible. Consequently, the specification stage, which could be facilitated by formal verification, becomes very important to the system life cycle. This paper presents a formal modelling approach, which may be used in order to better represent the reality of the system and to verify the awaited or existing system's properties, taking into account the environmental characteristics. For that, we firstly propose a formalization process based upon properties specification, and secondly we use Conceptual Graphs operations to develop reasoning mechanisms of verifying requirements statements. The graphic visualization of these reasoning enables us to correctly capture the system specifications by making easier to determine if desired properties hold. It is applied to the field of the Enterprise modelling.

***Keywords*:**

Enterprise Modelling, Conceptual Graphs, Properties Specification, Formal Verification

1. Introduction

Enterprise processes become increasingly complex taking into account not only classical technical and technological aspects but also social dependencies and the constant changing economic environment. Engineers need to model a given process in order to better understand it; the goal must be an ongoing process optimization to test a new control policy. Then the description and the verification and/or validation of each stage of an enterprise process specification become crucial, since they condition the quality, adequacy and efficiency of the services produced. Thus our approach contributes to its profitability by a central thread: modelling the process knowledge in a formal, provable, testable and re-usable form. This approach can be considered as a new way for Enterprise Modelling (Fox and Gruninger 1998) that aims to support understanding of what happens in an enterprise, support design and analysis of a business entity, improve knowledge level about the things of the enterprise and help to

[*] Corresponding author. Email: Bernard.Kamsu-Foguem@enit.fr

define more suitable models for decision-making activities both in the engineering and operation phases of the enterprise.

1.1 Modelling issues for integration and change management

Researchers draw distinctions between Systems Engineering and Requirements Engineering. Systems Engineering is “a discipline that concentrates on the design and application of the whole (system) as distinct from the parts. It involves looking at a problem in its entirety, taking into account all the facets and all the variables and relating the social to the technical aspect” (NAS 2004). Requirements Engineering is a branch of systems engineering that addresses translation of stakeholder needs into system requirements and facilitates the process by which the specification of systems and/or components satisfies those requirements (Young 2004). In order for efficient management of change to occur, enterprises have to cope carefully with the efficiency of the enterprise engineering or reengineering process. Business Processes modelling is an important part of the engineering effort, and modelling languages are essential components used at each step of Business Processes engineering methodologies (Petit *et al*. 2002, Girard and Doumeingts 2004). Models enable communication among the various people involved in the process in order to master the enterprise system’s complexity, to understand and analyse the situation, to re-engineer and to ultimately control or monitor the system. Moreover, enterprise modelling is a prerequisite to enterprise integration (Molina *et al*. 2004, Kosanke *et al*. 2003) because if parts of the enterprise are meant to communicate with each other, they should share some common models to enable better performance and quality results (see Vernadat (1996, 2002) for a detailed motivation for enterprise modelling). Also, enterprise systems and applications need to be interoperable in order to achieve seamless business interaction across organisational boundaries and thus realise networked organisations (ATHENA 2004, INTEROP 2003, Ducq *et al*. 2004).
Consequently, part of the re-engineering process efficiency concern lies in the efficiency of enterprise modelling process itself since models are used at all stages of this process. This need for efficiency imposes requirements (precision, easiness and expressiveness) on the languages used for producing models and (re-using knowledge about the existing system and generic knowledge reuse) on the process of modelling and designing systems.

1.2 Analysing Requirements Models

The primary measure of success of a requirement modelling approach is always done in terms of the kind of analysis and reasoning it offers. The analysis techniques can therefore be used to generate useful information from the models produced. Techniques such as goal driven approaches (like KAOS (Van Lamsweerde *et al*. 1998)), specification animation (Haumer *et al*. 1998) and the use of scenarios (like CREWS (Rolland *et al*. 1998, Sutcliffe 2002)) have received considerable attention in industrial applications, but they suffer from a lack of precision or exhaustiveness in analysis. Among many key elements in making the best possible decision in a requirement engineering project, formal approaches appear well suited in an early phase of a project life-cycle for checking global coherence and partial consistency between all the various requirements and specifications of different functions of the system (Morel *et al*. 2003). These *formal approaches* (Monin 2003), based upon mathematical constructs and set-theory notation, can be used to produce precise, unambiguous documentation, in which information is structured and presented at an appropriate level of abstraction.

Techniques such as model checking and theorem proving have a long tradition in formal specification languages like algebraic specifications (Ehrig and Mahr 1985), Z (Spivey 1992), CSP (Hoare 1985), Petri nets (Cortes *et al*. 2003), or temporal logics (Manna and Pnueli 1992). Despite of their relative maturity within software engineering research, there are very few practical applications to enterprise modelling and automation systems (Völker and Krämer 2002). Reasons for this include the complexity of formal specification techniques and the lack of training of enterprise engineers in applying them. Furthermore, there are also well-known limitations of formal verification such as the state-explosion problem within model checking.
This work attempts to bridge the semantic gap between the engineer and the formal specification by allowing results of an analysis to be easily reviewed for relevance. In such circumstance we wish to carry out exhaustive checks (models) of systems by using the elements (formal specification, formal reasoning, mathematical proof) of a formal approach (figure.1 adapted from (Grady 1997), summarizes this vision). In particular the principle of mathematical proof is to describe the properties (functional, behavioural and structural) needed in the form of a theorem and to show that it can be directly deduced from the specification.

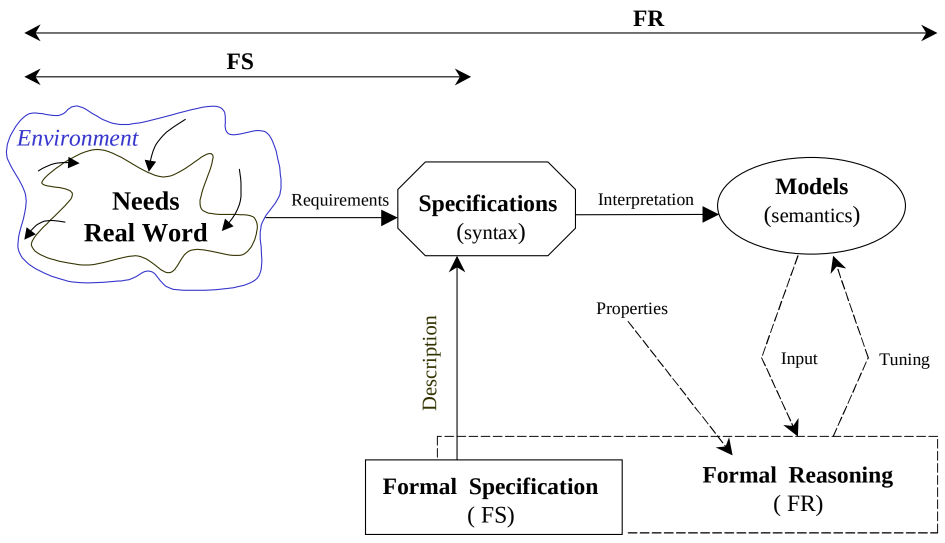


Figure 1. Example of formal approach

2. Formalization of knowledge: proposed approach

To formulate requirements in industrial systems, informal specifications are generally used, because they may seem, at first glance, easy to understand for the client. However, many problems appear (Robertson and Robertson 1999, Bray 2002), having sometimes heavy consequences on the development of the products:

– problems of communication between the client, the employee, and the manager (specifications can be interpreted in several ways);
– documents are not rigorous enough, inducing operational difficulties for later automatic processing;
– it is difficult to verify a specification, i.e to check that the requirements are correctly modelled.
– inherently fault-prone, which can prove costly if faults are discovered later in the integration and acceptance testing.

In front of such report, enterprise modelling has been developed for example, in the F3 project (Bubenko *et al*. 1994) to provide a set of models for understanding the

requirements and bridging the gap between ill-defined problems and application situations as well as to define requirements of information systems formally and precisely. In accordance with this point of view, the key idea of the work is to use a *formal approach* like an alternative by giving the user (modeller or engineer) the ability to use verification inside their task as a mean of reasoning and enriching his modelling or system specifications. This section presents the whole hypothesis allowing to develop a sequence of processing, as a solution to assist formalization. One major goal of the proposed formal specification method should be to facilitate the transmission of meanings through all the actors of an engineering systems process. For doing this, the conceptual graphs (CG) are the chosen formalism for knowledge representation, considering that they allow the representation of heterogeneous knowledge, a powerful structuring mechanism (Sowa and Zachman 1992), and they express meaning in a form that is logically precise, humanly readable and have a set of inference mechanisms (Chein and Mugnier 1992).
The intention is to formalize knowledge about system's requirements and to communicate the resulting patterns back to the engineers for eventual more advanced validation techniques and improvement. This way, we want to provide a framework within which people can specify, develop, and verify system models in a systematic, rather than ad hoc manner. Therefore, to answer the step of formalization, we propose a sequence to process the initial requirements, likely to provide a formal specification. The sequencing of the various stages (Natural Language (NL) requirements, properties description, knowledge representation and reasonings with graph operations) of this methodology (figure.2), represents the implementation of our objective, balanced by the whole of the hypothesis stated above.

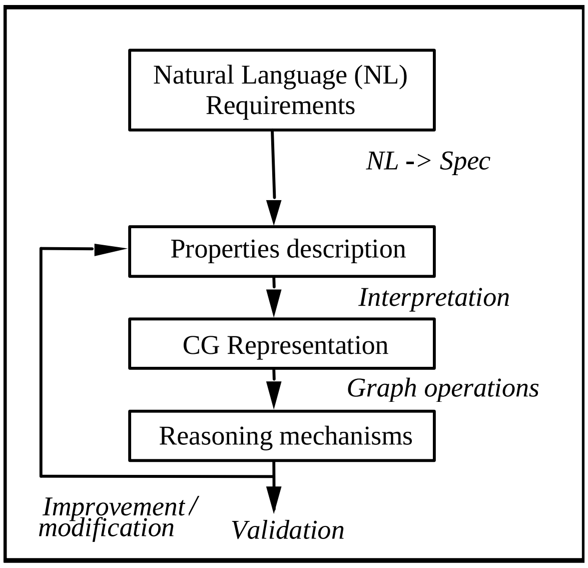


Figure 2. Methodology used for the construction of the formal specification

3. Properties description

3.1. Properties analysis

At the sight of the traditional problems arising at any requirements modelling and reasoning stage (heterogeneous knowledge, ambiguity, different points of view, hierarchical organisation, the system objective, etc), a formal model has been elaborated (Chapurlat *et al*. 2002) making possible the description of properties for requirements specification. A Property Reference Repository ((Kamsu 2004, Chapurlat *et al*. 2005) (not presented here) allows the task of selecting and specifying the relevant properties (of the pointed out system and of the model) to be simplified and thus accelerated. We argued that most requirements engineering problem domains are instances of a tractable set of object system models. Each model contains general features shared by all instances of that problem domain. For instance, one model of the NATURE project (Maiden and Hare 1998) contains general features of all resource hiring problem domains, examples of which are lending libraries, car rental and video hiring. Another contains general features of all object sensing problem domains. In our context, classifying properties is one approach for helping practitioners identify and specify common types of behaviours or situations. So, three kinds of properties are to be described, *axiomatic*, *system* and *model*:

Axiomatic Properties: These are facts, rules and laws which concern the application domain; the trivial knowledge about the application domain that the analyst can hardly invent. The axiomatic properties permit detailed reasoning about what is assumed about the domain, and they provide opportunities for requirements reuse within a domain. Described here, are the objects that will have to be modelled by the system and their attributes to fit one's surroundings. For instance, for a company that produces cars according to the orders received from customers, we have the following axiomatic properties:

- The enterprise consists of a physical system, a decision system and an information system. The control system, which is composed of the decision and information systems, controls the physical system and enables the enterprise as whole to reach its goals.
- The purchase of a new kind of machine by the manager *influences* the quality of products provided by the enterprise
- The growth in activities that are performed *implies* the increase in resource consumption.

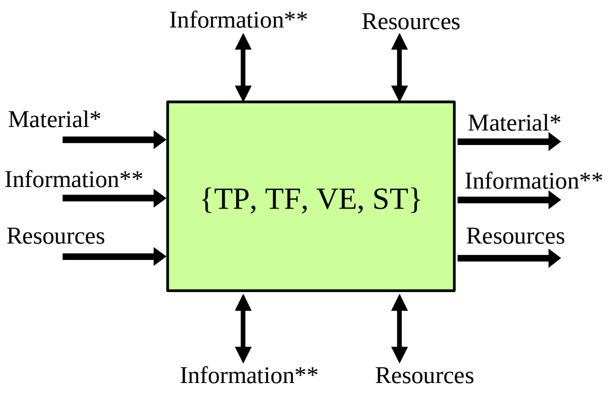
System properties: These properties rely on Systems Engineering (Martin 1997, Incose 2004) and express the characteristics of the target system (constraints, requirements, behavioural, functional and structural) and its assigned objectives. Examples for such properties are deadlock freedom, timing consistency and limited capacity resources. When developing a concurrent object-oriented application, deadlock freedom of the interaction is often a major requirement. Timing consistency is of importance for real-time systems. There, it must be assured that certain process are performed within a given pre-defined time span. Limited capacity resources are often a characteristic feature of embedded applications. Lamport in (Lamport 1977) described two others categories of system properties: safety (something bad never happens) and liveness (something good eventually happens).

Model properties: These properties characterize the features of the modelling language used (basic constructs, syntactic principles and semantic rules). Also, they enable the user to establish what to expect from the model: correctness, coherence, re-initialize state, parallelism, synchronization, sequence, bounded marking, cycle, temporal aspects, etc. This allows the user to translate some of the system's properties corresponding to requirements and expectations. For instance, if a highly serial process is operating too slowly to meet an impending deadline, the user may describe some actions on the model, such as to pipeline (i.e. release partial results to allow later tasks to start earlier) or parallelize to increase concurrency.

The user (modeller or engineer) can select, from this list of possible properties types, the ones that seem most important in his/her particular context. All these properties can be easily written in conceptual graphs as formal representation. Also, formalization of these properties as algebraic equations is reachable (Roussel *et al*. 2004), but the main weakness of such method lies in the need to manipulate algebraic statements that are not user friendly for enterprise engineers. Furthermore, to relate these classes of properties to specifications solutions, we developed an orthogonal classification of various properties that we linked to production system models to represent candidate specifications solutions for different requirements. To exploit the Property Reference Repository we developed computational models of analogical reasoning (Chapurlat *et al*. 2005) to retrieve types and characteristics of properties that matched a new application to enable reuse of knowledge about the problem domain and possible specifications system solutions to it.

3.2. Properties identification

At the methodological level, properties identification of complex systems is made by the following means: requirements analysis in order to identify system needs, test specifications to ensure completeness as well as functional and technical feasibility, statement description of the system's objectives and constraints according to knowledge of pattern-making methods, reading of technical documents and interview domain experts. Likewise, it is necessary to study the role and impact of domain knowledge (Jackson 1994). Since many new applications have the same requirements as earlier ones, one possibility is to create generic domain properties as templates for requirements of certain classes of applications (Maiden and Hare 1998). This facilitates reuse in properties identification by providing sets of predefined generic properties for developing system properties specification.
Furthermore, some international standards documents have been developed to aid properties critiquing as well as specification. For example, in the manufacturing domain, the ISO TC 184/SC5/WG1 Technical Report ISO 10314 (ISO 1990) provides a clear methodology (a list of structured questions that could be posed) to identify possible properties for areas of standards in support of integrated-shop-floor operations.



* Actions {TP, TF, VE, ST} on Material are defined only at level 1

TP = Transport
TF = Transform
VE = Verify
ST = Store

** Information include both control and data components. Horizontal information flows are restricted to data components.

Structured question: Should there be or are there already standards to relate:

$$\left\{\begin{array}{l}\text{Transform}\\ \text{Transport}\\ \text{Verify}\\ \text{Store}\end{array}\right\} \text{ with } \left\{\begin{array}{l}\text{Control information}\\ \text{Data}\\ \text{Material}\\ \text{Resources}\end{array}\right\} \text{ at the } \left\{\begin{array}{l}\text{Section}\\ \text{Cell}\\ \text{Station}\\ \text{Equipment}\end{array}\right\} \text{ level ?}$$

Figure 3 : Activity Model for Shop Floor Production (ISO 1990)

Complexity and complex systems, on the other hand, generally refer to a system of interacting units which displays global properties not present at the lower level. An emergent property cannot be understood simply by examining in isolation the properties of the system’s components. However, one can find it by investigating the nature of the rules governing interactions among system components (Ueda 2001). Others properties can be characterized, both formally and empirically, using Case-based Reasoning (reasoning technique that solves new problems by analogy to past problems, Shiu and Sankar 2004). It may be used to identify some similarities between systems used for different business purposes and to investigate their common properties and the general principles that underlie them (Watson 1997). As a result, Case-based reasoning systems can refer to a case base containing domain cases and find case that have characteristics similar to those of the current one. The similarities may cover the entire case or only certain points that led to a portion of the property. Cases can therefore be discovered that may support some portions of the current case while opposing other parts.

4. The tool of representation and reasoning: Conceptual graphs

The conceptual graphs are a language of knowledge representation, introduced by John Sowa in (Sowa 1984) and extended in (Sowa 2000, Baget and Mugnier 2002). Such language permits at the same time to define a vocabulary (i.e. ontology) and to use this vocabulary to conceptualize facts. Conceptual Graphs can be considered as a compromise representation between a formal language and a graphical language because it is visual and has a range of reasoning processes. Since, enterprise modelling demands correct models of the system and of its goals that are not easy to capture in an industrial context (Morel *et al*. 2001), we will use conceptual graph formalism to assist the requirements specification phase and express formally knowledge.

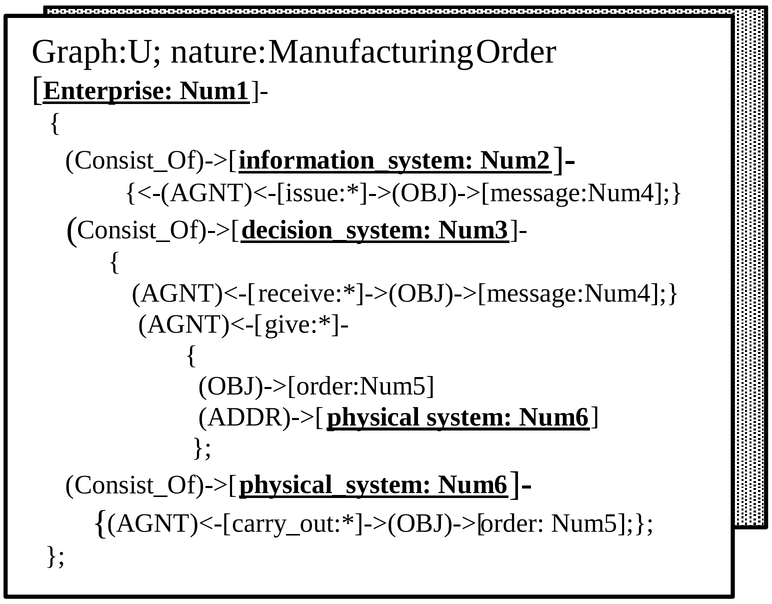
4.1 Formalism presentation

Definition: A simple conceptual graph is a finite, connected, directed, bipartite graph consisting of *concept* nodes (denoted as boxes), which are connected with conceptual *relation* nodes (denoted as circles). In the alternative linear notation, concept nodes are written within []-brackets while conceptual relation nodes are denoted within ()-brackets.

A **concept** is composed by a type and a marker [<type>: <marker>], for example [Resource: computer2]. The **type** of concept represents the occurrence of object class. They are grouped in a hierarchical structure called a concept lattice showing their inheritance relationships. The **marker** specifies the meaning of a concept by specifying an occurrence of the type of concept. They can be various natures, in particular individual, generic (symbol '*' within the marker), quantifiers, or sets (the latter by using {}-brackets within the marker). The term '{*}' denotes a set of zero or more elements, additional cardinality constraints can be expressed, for example, by '{*}@5' (set of five elements) or '{*}@>4' (set of more than four elements). It is also possible to pair the number with a unit of measure, for example the term '@96h' means ninety-six hours.

A conceptual **relation** binds two or more concepts according to the following diagram [C_1]←(*relation's name*)←[C_2] (means 'C_1 is in relation with C_2'). Each relation has a signature, which fixes its arity (the number of arguments it takes) and gives the maximum types of concept available, to which a relation of the type can relate. The sub-relation definition is sometimes necessary to provide more details in the semantic representation, and then a relation lattice is established.

Before representing knowledge with conceptual graphs, it is first necessary to determine an ontology (Guarino 1997) dedicated to what we need to represent. Thus, the formalism chosen here was adapted to take into account the modelling concepts of the enterprise (Berio 2003) by the construction of a support formed by a set of concepts, structured in a lattice and a set of relations between these concepts. The type used for concepts and relations must be declared or defined in our formal vocabulary where the terms may have associated constraints (e.g. signatures for the relation types) and definitions (e.g. definitions of necessary and/or sufficient conditions) and thus may be linked to other terms by different relations (e.g. given or calculated sub-assumption relations).



```
Graph:U; nature:ManufacturingOrder
[Enterprise: Num1]-
  {
   (Consist_Of)->[information_system: Num2]-
          {<-(AGNT)<-[issue:*]->(OBJ)->[message:Num4];}
   (Consist_Of)->[decision_system: Num3]-
        {
          (AGNT)<-[receive:*]->(OBJ)->[message:Num4];}
           (AGNT)<-[give:*]-
                {
                 (OBJ)->[order:Num5]
                 (ADDR)->[physical system: Num6]
                };
   (Consist_Of)->[physical_system: Num6]-
       {(AGNT)<-[carry_out:*]->(OBJ)->[order: Num5];};
  };
```

Figure 4. Graph *U* in the linear notation for specification of a 'Manufacturing Order'

The **Nested** Conceptual Graphs (Chein and Mugnier 1997) enables association of any concept node with a partial internal description. In addition nesting allows to create several representation levels, to organise these levels of detail into a hierarchy and thus to zoom on certain concepts by adding internal information to them. An important advantage of nested graph models is the option of partitioning the reasoning tasks into separate metalevel stages, each of which can be axiomatized in classical first-order logic. For that, it is defined a mathematical operator that translates conceptual graphs into formulas in the first-order predicate calculus (relations become n-ary predicates, concepts become unary predicates, individual markers become constants and generic markers become existentially quantified variables).

4.2. Conceptual Graph Operations

Conceptual graph operations provide a set of the reasoning mechanisms and define selected constraints of the graphs representing domain knowledge and facts. All these operations are mathematically founded both on logics (sound and complete) and graph theory (Sowa 1984, Chein and Mugnier 1992). Sowa has defined four elementary operations on the conceptual graphs, called *canonical formation rules*, which allows us to handle them easily and to derive canonically from other graphs: *copy, restriction, simplification* and *joint.* For instance, with *joint operation* two graphs having a common concept node can be merged to form one graph by sharing this common concept.

There exists another kind of handling operation called *projection* that is the fundamental element of a reasoning process for conceptual graphs. The projection search of a graph G in a graph H can be seen as the inclusion search of the information represented by G in H. This leads to a calculation in the specialization between the two graphs.

The example given in Figure 5, shows an application of the projection mechanism. This figure shows a situation in which :

- an activity called "to produce" has in input the rivet12 ,
- there is another activity called "to record" that has in output the order C17,
- with the request graph, we want to know if there exists an activity which has a given order in output.

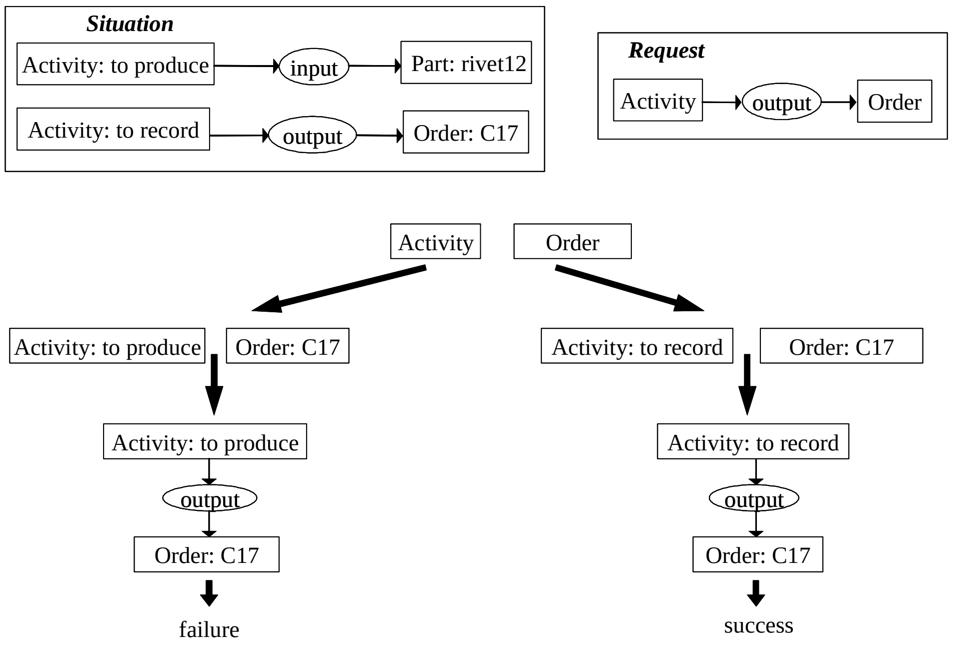


Figure 5 : Application of a projection

In this example, a request (in the form of a question graph) on the knowledge base consists in seeking if the graph question can be proved starting from the knowledge base. Beginning from a particular situation description and by taking as requests the states or the events, we can prove the presence of a property or the occurrence of a violated requirement. When a requirement is not satisfied and projection failure occurs, the reasoning steps can help diagnose their underlying causes, and suggest specific interventions for resolving them. Such reasoning with the projection (a graph matching operation) is interesting since the same language is used at interface and operational levels.

There exist two other kinds of possible use for projection in order to validate (constraints) or transform (graph rules) a graph in another one. Graphs operations like these may have been established by the automated reasoner of a knowledge-based system or by the engineer. Their explicit representation enables the analysis to give unambiguous information about the enterprise modelling process (causal and revision contexts), to draw the engineer's or machine agent's attention to data entries which do not exactly fit into the current view of the enterprise's situation (conflict contexts). Consequently, we are able to support the modelling process by providing knowledge that is highly enterprise-adapted, valid only for one particular part of enterprise (intra-enterprise similarity contexts, enterprise-specific heuristics).

4.3. Reasoning with graph rules

The conceptual graph rules (Bos *et al.* 1997, Kamsu *et al.* 2003) allow one to add new knowledge. The graph rule is composed of a hypothesis and a conclusion, and is used in the following classical way: given a simple graph, if the hypothesis of the rule projects to the graph, then the information contained in the conclusion is added to the graph. Rules are split into *static rules* and *dynamic rules*:

- ***Static rules*** express some immutable domain laws, and their uses complete words descriptions. In order for an enterprise model to support common-sense query processing, it must provide a set of rules of deduction, known as axioms. Here, let 'works-for' be a binary relation, whereby we require an axiom stating that 'works-for' is transitive :

  x works-for ***y*** AND ***y*** works-for ***z*** IMPLIES ***x*** works-for ***z***.

- ***Dynamic rules*** define possible transitions from one word to another. The successor of a valid word is obtained by a single application of a transformation rule on this word. From this perspective, we specify some dynamic rules in order to describe precisely how the actors will interact with the enterprise. Some new information can be introduced: the actions performed by the actors who interact with the enterprise, the operations executed in response to an action by the enterprise and the observable states (by some actors) of the enterprise. Like that, each dynamic rule is defined as a triplet *action[condition]/response*. In the case of car factory, the example (figure 6) is as follows :

  'A customer places an order for a car [car available] / the enterprise does home deliveries within 96 hours'.

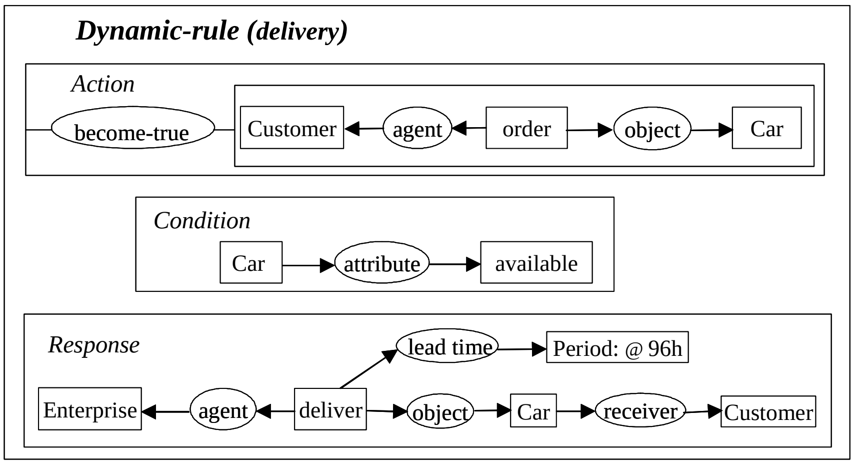


Figure 6: Example of a dynamic rule

4.4. Graph constraints

A *constraint* defines conditions for simple graph to be valid (Baget and Mugnier 2002). It is composed of a *conditional* part and a *mandatory* part. The condition must be a simple graph. In particular, a condition can be an *empty graph*. Roughly speaking, a graph satisfies a constraint if for every projection of its conditional part, its mandatory part also projects to the graph. We consider positive and negative constraints. A positive constraint expresses a property such as '*if information A is present, then information B must also be present*'. For example, any failing resource must be repair or must be substitute for new one. A negative constraint expresses a property such as '*if information A is present, then information B must be absent*'. For example, every operational process must not have two incompatibles activity or a repairing failure context must be viewed as being inconsistent with some diagnosis. This kind of property express that data are viewed to be inconsistent or that some activities may raise conflicts. After possible conflicts have been identified, the next step is to use the information stored in the Property Reference Repository (section 3) in order to find ways for avoiding or detecting the conflicts. Although conflicts usually are resolved after some time (by having more information about an enterprise model), they should be represented for a better understanding of the decision process.

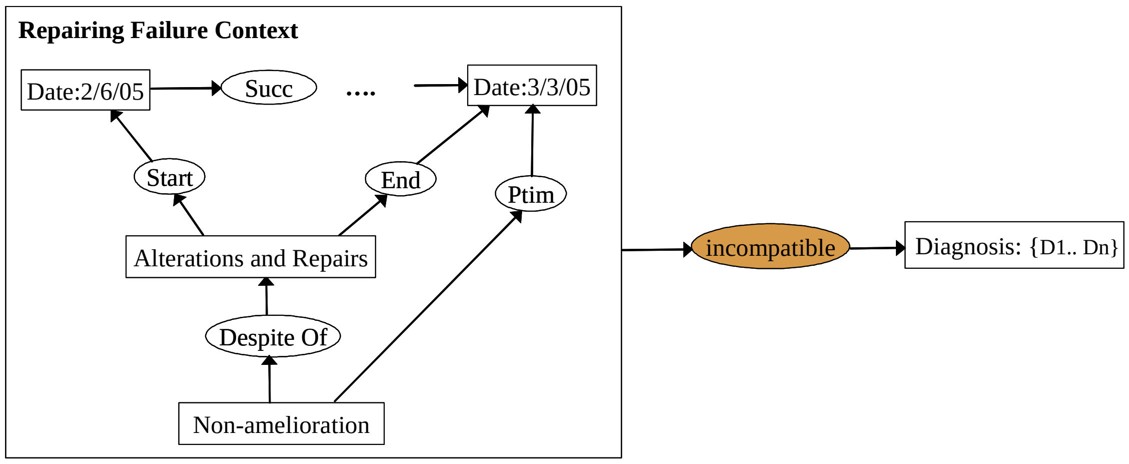


Figure 7: Example of a graph constraint

In fact, the constraints can be applied to express mathematical coherence in the case of model analysis or to ensure and/or restore consistency of the systems specifications. Indeed, some of the systems specifications based upon users needs might be constraining from a technical and/or a domain point of view. Hence, constraints are used to check that a specification provides an accurate account of stakeholder requirements.

5. Case Study: a product development process

In this section, we will demonstrate how the approach described in this paper can be used to systematically specify and verify properties (or point out possible gaps) in an experimental case of a business process. The approach begins with a construction of abstract description (i.e. a model of the product development process) that is amenable to interpretation. The construction of such model may be done with a modelling language like UEML (Berio 2003). Particularly, the model of the product development process typically includes a “design product” activity, followed by a “manufacture product” activity, which, in turn, is followed by a “deliver product” activity. Figure 8 depicts a simplified but accurate model of this process, based on the descriptions contained in (Kamsu 2004). The model consists of boxes, which describe process activities, and lines, which describe various dependency relationships, that is, constraints that must hold true in order for the process to succeed.

This model has been characterized by several properties:

- If one then uses a type of manufacturing activity, Then the material used and the geometry of the produced objects have indissociable features of this activity.
- The output of the “design product” activity must be consistent with the capabilities of the “manufacture product” activity.
- The shape attributes of the output and input of the “deliver product” activity must be equal.
- If several activities are causing wasteful overheads by frequently trading the use of a scarce shared resource, Then change the resource sharing policy such that each activity gets to use the resource for a longer time.

- The deadlock situations (where several activities are each waiting for another one to do something) are forbidden.
- The resource poaching situations (wherein high-priority activities are unable to access needed resources because these resources they have already been reserved by lower priority activities) are forbidden.

These properties can be applied in order to help design effectual new processes. They can also be a helpful knowledge when verifying the model of studied business process. Explicitly describing the properties is a necessary precondition not only for verifying requirements, but also for resolving conflicts between stakeholders (person and team that are responsible for or in some way have a vested interest in the requirement or product under consideration).

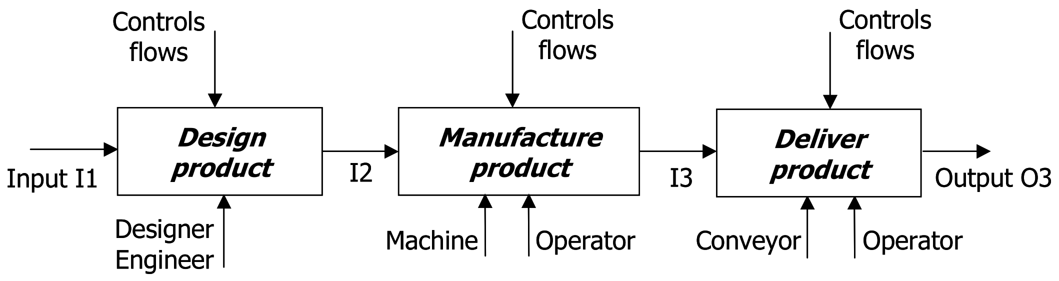


Figure 8: A model of the "product development" process

5.1. Model verification

One central goal of model-based development is to enable analysis of the system, thus ensuring the quality of the system already on the model level. That is, we want to reason about certain properties of the system prior to the construction of the implementation. The purpose of verification is to prove that, under a certain set of assumptions, a system will adhere to a certain set of properties, for example the design requirements. For verification of properties, first a suitable formal verification tool has to be chosen capable of verifying the aspects associated to the property. Lightweight formal methods (Easterbrook *et al.* 1998, Jackson 2001) (for instance a formal conceptual modeling) show significant promise in this context, as they offer a way of uncovering major errors without the burden of full proofs of correctness.
We will concentrate on model-based verification and the property verification is not directly performed on model level. During property verification, the model is translated into a suitable conceptual graph. In order to build the conceptual graph corresponding to a given enterprise process model, we have developed and implemented an algorithm (Kamsu 2004) allowing this translation, the important characteristics of the same is described as follows: the concepts of process model are translated into the conceptual graph according to their respective marker (individual, generic or variable) and relations are translated in terms of typed relations, pre-conditions and post-conditions. Since the transformation is isomorphic, we assume that the translation is semantic-preserving. As a consequence, if the property is not fulfilled, we can conclude that the model itself does not fulfil the property. The following section offers a view of this kind of verification, wherein the reasonings are modelled using graph theory, and the anomalies are defined in graph-theoretic terms.

5.2. Proof of properties

Let *G* is the graph representing a model, *P* is the query graph representing the property to prove, R is a set of implicit knowledge rules and *C* be a set of constraints depending from the domain. *P* is deduced from (*G*, *R*, *C*) if it is possible to obtain a valid graph *G'* by a sequence of immediate transformations on *G*, such that *P* can be projected into *G'*. In this case, the property P will be verified. In other case, conceptual graph theory offer some means allowing us to establish what are the possible causes of non verification of P. This permits to highlight some defaults or mistakes and then to improve the model. This way, conceptual graphs provides inference mechanisms for proving properties by using projection, rules and constraints. These demonstrative abilities make verification knowledge possible. For example, if we consider the product development process, we want to verify the following property P_1 :

P_1 *'If one then uses a type of manufacturing activity then the material used and the geometry of the produced objects have indissociable features of this activity'.*

For the purpose of this verification task, one may use the negative constraint *Nc* and the rule R_1 presented below:

Nc 'Two equivalent manufacturing activities must provide in output some identical products'

R_1 *'If two given products have different geometrical attributes then they are different'.*

We will use the **proof by reductio ad absurdum** that is a method of proof which proceeds by stating a proposition and then showing that it results in a contradiction, thus demonstrating the proposition to be false. This proof by reductio ad absurdum of P_1 in natural language is as follows: supposing that in the production system, there are two products *x* and *y* which are coming from two equivalent manufacturing activities, with different geometrical attributes. The previous rule R_1 tells us that the product *x* is different to the product *y*. Consequently, there are two equivalent manufacturing activities that produce different products. This result breaks the rule N_C and belies our starting hypothesis, so it all goes to prove property P_1.
The sequences of proof are formalized in conceptual graphs as follows:
- The starting hypothesis of proof is represented by the conceptual graph *Gh* in figure 9.

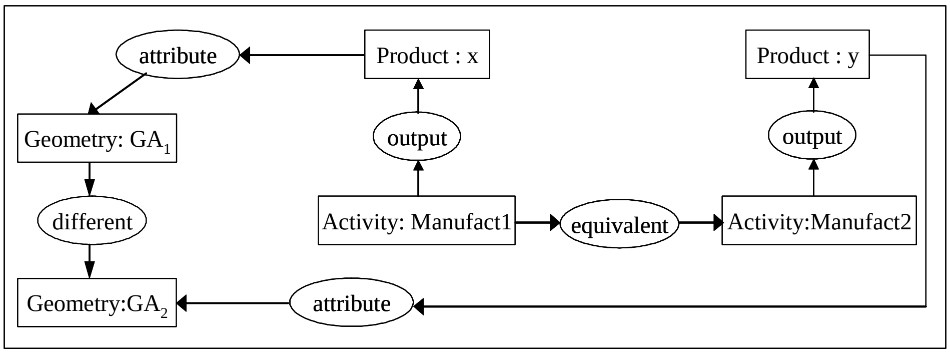


Figure 9 : Graph Gh of the starting hypothesis to proof

- The result of the application of rule R_1 to graph *Gh* is represented by the conceptual graph *Gc* in figure 10.

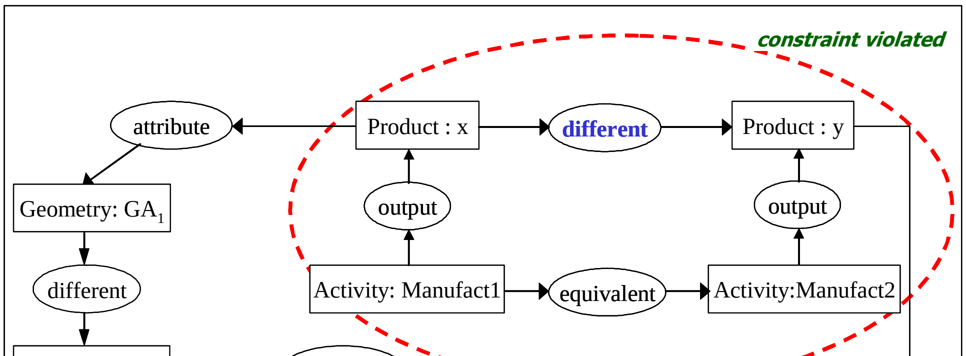


Figure 10: Graph Gc resulting from the application of rule R_1 to graph Gh reveals one contradiction

The condition of negative constraint Nc is the empty graph, which can be projected into Gc. And this constraint is violated since there exists a projection of Nc as a whole into Gc. It is a contradiction in terms, so the proof of property P_1 is established.
The practical interest of this property is that the user wishing to produce parts with specific features, will be able to choose the adequate workshop to perform his task in relation with the working environment. A major asset is that the proposed approach enables the end-user (an engineer or even an expert) to follow reasonings step by step, directly on his own model-building.
In this manner, with the graph operations, we are able to verify a model through mathematical argument (proving properties of specification) and then we are more likely to detect problems at an early stage of system development. Indeed, the process of constructing proofs can help us to understand the requirements upon a system, and can assist us in identifying any hidden assumptions (Chapurlat *et al*. 2003). These proofs guarantee that the system will always satisfy its properties which are the facts, domain rules and needs. Hence proofs at the specification stage can make a significant contribution to the quality of the system.

6. Conclusion

In this paper we propose a formal approach for modelling and analysing requirements. For this, we firstly represent the requirements in the form of precise specification of the system's properties with the description of its externally known characteristics. Secondly these specifications are translated into Conceptual Graphs which provide a formal ontology for domain modelling and graph operations that are logically founded. These graph operations allow us to support formal reasoning to verify the internal consistency of specifications, and to validate a formal specification against an informal description of the system, by proving properties which it should exhibit. The verification process of desirable requirements of the system to be developed is done with the CoGITaNT software (Genest 2005) (a C++ library for developing conceptual graphs applications, and a conceptual graph editor) which guarantees that a model of the system will always satisfy its properties. Our work can be considered as an innovative approach in the sense that it allows the user (modeller or analyst) to follow reasoning processes in a graphical way. Finally, the main contribution of the proposed approach is that it helps in clarifying requirements expected of systems, in improving the rigour of the analysis performed and in making the reasoning steps explicit.

7. Related and Future Works

Usually, some existing techniques are applied to requirements analysis. For example, although goal-based methods, like KAOS approach (Van Lamsweerde *et al*. 1998), has proved to be useful for specifying purposeful systems, practical experience shows that there are still a number of difficulties (eliminating uninteresting and spurious goals is difficult (Potts 1997), the goal discovery process is not straight-forward (ELEKTRA 1998, Sutcliffe 2002), etc.). On the other hand, the proposed approach helps in identifying the needed properties and the graph-based approach allowing to prove them, in an organised and systematic way in order to ensure the adequacy of the future system with a given set of objectives. Nevertheless it is true that Conceptual Graphs are not intended for the modelling of non-functional properties, such as usability, efficiency and reliability. Neither is it intended for the description of temporal or concurrent behaviour. Indeed, as a result other formal languages (Statecharts (Harel and Politi 1998), Albert II (Dubois *et al*. 1994), Method B (Abrial 2005), I* (Chung *et al*. 2000), etc.) which can be used in combination (Paige 1998) with Conceptual Graphs, would be better suited for such modelling.
Our system will be presented to the end user in the industry like a case tool driven approach, with the conceptual graph "engine". Such engine may be hidden from the non-technical end user, since there are some good translations from conceptual graphs into natural language (Diallo 2000, Dieng and Corby 2005), which provides rich representations (narrative descriptions of contexts) that non-technical end user find appealing. Accordingly, and again similarly to work done by (Gouyon *et al* 2004) in automation engineering context, our graph-based approach would facilitate the generation of purposeful system requirements leading to better quality systems (Rolland and Prakash 2000) and therefore highlights a pragmatic use of formal analysis within an engineering process.
The current application of our research is the integration into a connectionist-symbolic framework (Vilhelm *et al.* 2000) aiming to support the decision-making process in medical domain and future direction are related to the capitalization and the diffusion of knowledge from experience feedback (Hermosillo *et al.* 2005) in an organization.

Acknowledgments

This work was partially carried out with the collaboration of the Montpellier Laboratory of Computer Science, Robotics, and Microelectronics (LIRMM). Professor François Prunet provides advice and many helpful comments on an earlier version of the work. Special thanks to the anonymous referees for their interesting suggestions. I am grateful to Professor Blaise Nsom Eyenga for many simulating discussions on the research in general.

8. References

Abrial, J-R. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 2005, ISBN : 0521021758.

ATHENA. Advanced technologies for interoperability of heterogeneous enterprise networks and their applications, FP6-2002-IST-1, Integrated project description of work; 2004.

Baget, J-F. and Mugnier, M-L. Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints. *Journal of Artificial Intelligence Research* (*JAIR*), vol. 16, 2002, pages 425-465.

Berio, G., Requirements analysis: initial core constructs and architecture, Deliverable D3.1 of the UEML Project, Unified Enterprise Modeling Language Thematic Network, IST-2001-34229, May 2003, document available from the UEML portal at: http://www.ueml.org..

Bos,C., Botella, B. and Vanheeghe, P. Modelling and Simulating Human Behaviours with Conceptual Graphs. Proceedings of *Fifth International Conference on Conceptual Structures, ICCS'97*, Seattle, Washington, USA, August 3-8, 1997. Lecture Notes in Computer Science 1257, Pages 275-289, Springer 1997.

Bray, K. An Introduction to Requirements Engineering, Addison-Wesley, August 2002.

Bubenko J., Rolland C., Loucopoulos P., and De Antonnellis V. "Facilitating Fuzzy to Formal Requirements Modelling", Proc. Int. Conf. on Requirements Engineering (ICRE), Colorado Springs, USA, 1994.

Chapurlat,V., Kamsu Foguem, B. and Prunet, F. A Property Relevance Model and associated Tools For System Life-Cycle Management. *In 15th IFAC World Congress on Automation Control (B'02*), Barcelona, SPAIN, 21-26 July 2002.

Chapurlat,V., Kamsu Foguem, B. and Prunet, F. Enterprise Model Verification and Validation: an Approach. *IFAC Annual Review in Control*, 2003, volume 27, n° 2, pp. 185-197. Publisher: Elsevier Science.

Chapurlat,V., Kamsu Foguem, B. and Prunet, F. A Formal Verification Framework and Associated Tools for Enterprise Modeling: Application to UEML, Computers in Industry, In Press, Corrected Proof, Available online 7 September 2005.

Chein, M. and Mugnier, M-L. Conceptual Graphs: Fundamental Notions. Artificial Intelligence Review, volume 6-4, pages 365-406, 1992.

Chein, M. and Mugnier, M-L. Positive Nested Conceptual graphs. In D. LUCKOSE, H. DELUGAH, M. KEELER, L. SEARLE, and J.F. SOWA, publishers. Conceptual Structures: Fulfilling Peirce’s Dream, *proceedings of the fifth International Conference on Conceptual Structures (ICCS’97*), Seattle, USA, volume 1257 of Lecture Notes in Artificial Intelligence, pages 95-109, Springer Verlag, 1997.

Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Boston: Kluwer Academic Publishers, 2000.

Cortes, L.A., Eles, P. and Peng, Z. Modeling and formal verification of embedded systems based on a Petri net representation. Journal of Systems Architecture, Volume 49, Issues 12-15, December 2003, Pages 571-598.

Diallo, D. Assistance to validation through paraphrasing of formal specification written in B. (in French). PhD thesis, University of Nantes, 2000.

Dieng, R. and Corby, O. « Conceptual Graphs for Semantic Web Applications », In: Proc. of the 13th Int. Conference on Conceptual Structures (ICCS'2005), F. Dau, M. L. Mugnier, G. Stumme (editors), Kassel, Germany, July 17-23, 2005, Springer-Verlag, LNAI 3596, pp. 19-50, 2005.

Dubois, E., Du Bois, P., Dubru, F. and Petit, M. Agent-Oriented Requirements Engineering: A case Study using the ALBERT Language, Proc. of the *Fourth International Working Conference on Dynamic Modelling and Information System* - DYNMOD-IV, A. Verbraeck, H.G. Sol, and P.W.G. Bots (editors), Noordwijkerhoud, The Netherlands, september 28-30, 1994.

Ducq V., Chen D., and Vallespir, B. Interoperability in enterprise modelling: requirements and roadmap. Advanced Engineering Informatics, Volume 18, Issue 4, October 2004, Pages 193-203.

Easterbrook S. M., Lutz, R. Covington, R. Kelly J., Ampo, Y. And Hamilton, D. "Experiences Using Lightweight Formal Methods for Requirements Modeling". IEEE Transactions on Software Engineering, Special Issue on Formal Methods in Software Practice, vol. 24, (1), 1998.

Ehrig, H. and Mahr, B. Fundamentals of algebraic specifications 1: Equations and initial semantics, volume 6 of EACTS Monographs on Theoretical Computer Science. Springer, Berlin, 1985.

ELEKTRA consortium. DEMETRA: System Design Specification for PPC, ELEKTRA deliverable, March 1998.

Fox, M.S. and Gruninger, M. "Enterprise Modelling", AI Magazine, AAAI Press, Fall 1998, pp. 109-121.

Genest, D. CoGITaNT Version-5.1 – Reference Manual. Document available online at: http://cogitant.sourceforge.net.

Girard, P. and Doumeingts, P. GRAI-Engineering: a method to model, design and run engineering design departments. *International Journal of Computer Integrated Manufacturing*, Volume 17, Number 8, Pages: 716-732 / December 2004 (Taylor & Francis: London).

Gouyon, D. Petin, JF. and Gouin, A. Pragmatic approach for modular control synthesis and implementation. International Journal of Production Research, vol. 42, n° 14, pp. 2839-2858, ISSN 0020-7543, July 2004.

Grady, J. System Validation and Verification, CRC Press, 1997.

Guarino, N. Understanding, Building, and Using Ontologies: A Commentary to "Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber, and Wielinga. International Journal of Human and Computer Studies, Vol. 46, 1997, pp. 293-310.

Harel, D. and Politi, M. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw-Hill, 1998, ISBN 0-07-026205-5.

Haumer, P., Pohl, K. and Weidenhaupt, K. *Requirements Elicitation and Validation with Real World Scenes*. IEEE Transactions on Software Engineering, Vol. 24, No. 12, Special Issue on Scenario Management, December 1998.

Heitmeyer, C. L., Jeffords, R. D. and Labaw, B. G. Automated Consistency Checking of Requirements Specifications. IEEE Transactions on Software Engineering and Methodology, vol. 5, no 3, pp. 231-261, 1996.

Hermosillo Worley, J., Rakoto, H., Grabot, B. and Geneste, L. A competence approach in the experience feedback process, in "Integrating Human Aspects in Production Management", Series: IFIP International Federation for Information Processing, Vol. 160, Zulch, Gert; Jagdev, Harinder S.; Stock, Patricia (Eds.) 2005, Kluwer Academic Press.

Gramlich, B. Strategic Issues, Problems and Challenges in Inductive Theorem Proving. Electronic Notes in Theoretical Computer Science, Volume 125, Issue 2, 15 March 2005, Pages 5-43.

INCOSE SE Handbook Working Group, System Engineering Handbook, A « How To » Guide For All Engineers, July 2004.

INTEROP. Interoperability research for networked enterprises applications and software, network of excellence, proposal part B, April 23 2003.

Hoare, C. A. R. Communicating Sequential Processes. Prentice Hall, 1985.

ISO 10314. 'Reference Model for Shop Floor Production Standards', Technical report 10314, Part 1, ISO TC 184/SC5/WG1 N126 and Part 2, ISO TC 184/SC5/WG1 N160. 1990.

Jackson, M. Problems, Methods and Specialisation. *Software Engineering Journal*, Volume 9, Number 6, pages 249-255, November 1994; edited and abridged in IEEE Software ,Volume 11, Number 6, pages 57-62, November 1994.

Jackson, D. Lightweight Formal Methods. Proceedings of International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001.

Kamsu Foguem, B. Complex Systems Properties Modelling and Verification: Application to Enterprise Process Analysis (in French). PhD thesis, University of Montpellier II, July 2004.

Kamsu Foguem, B., Chapurlat,V. and Prunet, F. Enterprise Model Verification : a Graph-based Approach. Symposium on Discrete Events in Industrial and Manufacturing Systems; *CESA'03, IEEE/SMC multiconference on Computational Engineering in Systems Applications*. Lille, France, July 2003.

Kosanke, K., Jochem, R., Nell, J.G. and Ortiz Bas, A. *Enterprise Inter- and Intra-organisational Integration - Building an International Consensus*, Kluwer Academic Publishers, 2003, ISBN 1-4020-7277-5.

Lamport, L. Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering, SE-3(2):125-143, March 1977.

Maiden, N.A.M. and Hare, M. Problem domain categories in requirements engineering. *International Journal of Human-Computer Studies*. 49(3): 281-304, September 1998.

Manna, Z. and Pnueli, A. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer-Verlag, 1992.

Martin J. N., System Engineering Handbook, CRC press, 1997.

Molina, A., Panetto, H., Chen, D., Vernadat, F. and Whitman, L. Enterprise Integration and networking: Milestone Report, TC 5.3 Enterprise Integration and Networking, Proceedings of ICEIMT2004, *International Conference on Enterprise Integration and Modeling Technology*, Canada, 2004.

Monin, J-F. *Understanding Formal Methods*. Springer-Verlag, January 2003.

Morel G., Pétin J. F., Lamboley P. Formal specification for manufacturing systems automation; 10th IFAC/INCOM Symposium, Vienna, Austria, 2001.

Morel G., Panetto H., Zaremba M. and Mayer, F. Manufacturing Enterprise Control and Management System Engineering: paradigms and open issues. *IFAC Annual Review in Control*, Volume 27, Issue 2, 2003, Pages 199-209.

NAS (National Airspace System). NAS System Engineering Manual Version 3.0. Published in September 2004. Document is available to http://www.faa.gov/asd/SystemEngineering/

Paige, R.F. Heterogeneous Notations for Pure Formal Method Integration. *Formal Aspects of Computing* 10(3): 233-242, Springer-Verlag, June 1998.

Petit, M. and Doumeingts, G., Enterprise modelling state of the art, Deliverable D1.1 of the UEML Project, IST-2001-34229, November 2002. Document available at UEML portal: http://www.ueml.org.

Potts, C. Fitness for use : the system quality that matters most. Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97, Barcelona, pp. 15-28, June 1997.

Ueda, K. Synthesis and Emergence - research overview. Artificial Intelligence in Engineering, 15, pp 321-327, published by Elsevier Science Ltd, 2001.

Robertson S. and Robertson J. Mastering the requirements process. Addison-Wesley, 1999.

Rolland C., Ben Achour C., Cauvet C., Ralyté J., Sutcliffe A., Maiden N.A.M., Jarke M., Haumer P., Pohl K., Dubois E. and Heymans P., "A Proposal for a Scenario Classification Framework". Requirements Engineering Journal, Vol; 3, No. 1, pp. 23-47, 1998.

Rolland C., and Prakash N. From Conceptual Modelling to Requirements Engineering. Special Issue of Annals of Software Engineering on "Comparative Studies of Engineering Approaches for Software Engineering", 10,pp.151-176, 2000.

Roussel J. M., Faure J. M., Lesage J. J. and Medina A. An algebraic approach for dependable logic control systems design. International Journal of Production Research; vol. 42, n° 14, pp: 2859-2876, July 2004.

Shiu, S. and Sankar, K.P. Foundations of Soft Case-Based Reasoning. *Wiley Series on Intelligent Systems*. 274 pages, Publisher: Wiley-Interscience (March 5, 2004), ISBN: 0471086355.

Sowa, J.F. *Conceptual structures: information processing in mind and machine*. New York (U.S.A.): Addison-Wesley, 1984.

Sowa, J.F. and Zachman, J.A.., Extending and Formalising the Framework for Information Systems Architecture, *IBM Systems Journal*, volume 31, numéro 3, pages 590 - 616, 1992.

Sowa, J.F. *Knowledge Representation: Logical, Philosophical, and Computational Foundations,* Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

Spivey, JM. *The Z Notation: A Reference Manual* (2nd Edition). Prentice Hall, 1992.

Sutcliffe, A. User-Centred Requirements Engineering: Theory and Practice. Springer, ISBN: 1852335173, 2002.

Van Lamsweerde, A., Darimont, R. & Letier, E. Managing conflicts in goal-driven requirements engineering. IEEE Transactions on Software Engineering, 24(11): 908-926, 1998.

Vernadat, F. B., Enterprise Modeling and Integration: Principles and Applications, (London: Chapman & Hall), 1996.

Vernadat, F.B. UEML: towards a unified enterprise modelling language. International Journal of Production Research, Volume 40, Number 17, Pages: 4309 – 4321 / November 20, 2002.

Vilhelm, C., Ravaux, P., Calvelo, D., Jaborska, A., Chambrin, M-C., and Boniface, M. Think!: a unified numerical-symbolic knowledge representation scheme and reasoning system. Artificial Intelligence, Vol. 116, no. 1-2, pp. 67 - 85, January 2000.

Völker, N. and Krämer, B.J. Automated verification of function block-based industrial control systems. Science of Computer Programming, Volume 42, Issue 1, January 2002, Pages 101-113.

Watson, I. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann: San Mateo, CA, US, 1997.

Young, R. The Requirements Engineering Handbook. Artech House Publishers, ISBN 1580532667, 2004.