A formal verification framework and associated tools for Enterprise Modeling: Application to UEML

V. Chapurlat [a,*], B. Kamsu-Foguem [a], F. Prunet [b]

[a] *LGI2P-Laboratoire de Génie Informatique et d'Ingénierie de Production, Site EERIE de l'Ecole des Mines d'Alès, Parc Scientifique George Besse, 30035 Nîmes Cedex 1, France*

[b] *LIRMM, Laboratoire d'informatique, de Robotique et de Microélectronique de Montpellier, 165, rue ada, 34065 Montpellier Cedex 5, France*

Abstract

The aim of this paper is to propose and apply a verification and validation approach to Enterprise Modeling that enables the user to improve the relevance and correctness, the suitability and coherence of a model by using properties specification and formal proof of properties.

Keywords: Enterprise Modeling; Verification; Validation; Property; UEML

1. Enterprise Modeling versus enterprise model verification: problematic

A model is a representation of reality that enables a user to understand a system or phenomenon better, to evaluate some of its characteristics, and to share and argue opinions about it with other users. Usually a model is used to support a decision-making process to determine what actions (design, improvement or control) have to be carried out on the system concerned.

Enterprise Modeling (EM) [1] is defined by [2] as *the art of externalizing enterprise knowledge which adds value to the enterprise or needs to be shared. It consists of making models of the structure, behavior and organization of the enterprise*. Several approaches, methods, reference models, architecture models, norms and tools have been defined over the last 20 years.

A broad study of these is given in [19] and some of the more important ones are described in Refs. [3,4]. We can cite, for example, the ICAM Definition (IDef) family which includes several modeling languages (Idef-0 and other) [5], CIM Open System Architecture (CIMOSA proposing different views of the enterprise: functional, informational, resource and organization) [6], GRAI and GRAI-GIM (GRAI Integrated Methodology is a methodology for the design and analysis of production systems, and more particularly decision systems, in an enterprise) [7], Process Specification Language (PSL) [8,9], TOronto Virtual Enterprise (TOVE) [10], Business Process Modeling Language (BPML) [11], Aris [12], some reference models, such as Purdue Enterprise Reference Architecture (PERA) [13] or Generalized Enterprise Reference Architecture and Methodology (GERAM) [14,15] and some standards [16–18,63,64].

Finally, several tools (ARIS ToolSet, MEGA Process, FirstSTEP, MooGo, Graisoft 1.0, etc.) supporting some of these approaches provide modeling and analysis functionalities enabling the behavior of a part of an enterprise to be represented and investigated.

All of these approaches and tools offer modeling concepts, relations between concepts and constructs that usually highlight the relevant entities that make up an enterprise. Some of these concepts and relations are common and focus on the same entity as an activity or a process. However, they are defined differently from one language to another. For example, a GRAI activity and a PSL activity do not have the same semantic. On the other hand, some languages propose particular concepts or constructs which do not exist in another languages but are necessary to build models responding to a particular use.

This results a ''Tower of Babel'' [2] situation in which it is necessary to adopt a consensus and to determine a set of core concepts and constructs for Enterprise Modeling. The goal is not to define a unique modeling language that will replace all

* Corresponding author. Tel.: +33 466 703 866; fax: +33 466 387 074.
E-mail address: Vincent.Chapurlat@ema.fr (V. Chapurlat).

the others. The goal is to define a pivot modeling language allowing to improve the communication between the existing languages without loss of information and semantics, and to define a set of common concepts which have to be taken into account in the development of future modeling languages. Researchers have focused on unified languages, such as PSL (dedicated to the representation of manufacturing systems) or Unified Enterprise Modeling Language (UEML) which is dedicated to the representation of business processes [19–22].

Finally, with these modeling languages available, any obtained model *can be validated and checked for rigor and robustness* [23]. Indeed, it is necessary to guarantee the user a given level of confidence about the suitability, correctness, relevance and fidelity of each model before using it. In other words, it is necessary to integrate concepts and mechanisms into modeling languages or modeling tools that support or facilitate verification and validation (V&V) tasks:

- *Verification* must *confirm* [24] *by examination and provision of objective evidence that specified requirements have been fulfilled*, that is to say to respond to the question ''Is the model well formed?'' This naturally involves checking the syntax, but also verifying the semantic relevance of the model (the model respects the language construction and behavioral rules and the concepts used are clearly identified and correctly interpreted).
- *Validation* must *confirm* [24] *by examination and provision of objective evidence that the particular requirements for a specific intended use are fulfilled*, that is to say ''Is it the intended model?'' or ''Is the model faithful to reality?'' This involves assuming that the behavior of the model is equivalent to the behavior of the system, if it exists, or to the requirements, taking into account the modeling hypotheses.

Verification and validation need to provide rigorous arguments in order to convince users of the correct functioning and reliability of a model and of model-based systems before using them [25,26]. It also ensures the coherence between the different models of a given enterprise, thus improving communication and exchange among the different users, or

Table 1
Verification and validation techniques (V&V) [29]

Category	Verification and validation techniques (V&V)	
Informal	Audit	Desk checking
	Face validation	Inspections
	Reviews	Turing test
	Walkthroughs	
Static	Cause–effect graphing	Control analysis (calling structure; concurrent process; control flow; state transition)
	Data analysis (data dependency; data flow)	Fault/failure analysis
	Interface analysis (model interface; user interface)	Semantic analysis
	Structural analysis	Symbolic evaluation
	Syntax analysis	Traceability assessment
Dynamic	Acceptance testing	Alpha testing
	Assertion checking	Beta testing
	Bottom-up testing	Comparison testing
	Compliance testing (authorization; performance; security; standards)	Debugging
	Execution testing (monitoring; profiling; tracing)	Fault/failure insertion testing
	Field testing	Functional (black-box) testing
	Graphical comparisons	Interface testing (data; model; user)
	Object-flow testing	Partition testing
	Predictive validation	Product testing
	Regression testing	Sensitivity analysis
	Statistical techniques	Special input testing (boundary value; equivalence partitioning; extreme input; invalid input; real-time input; self-driven input; stress; trace-driven input)
	Structural (white-box) testing (branch; condition; data flow; loop; path; statement)	Sub-model/module testing
	Symbolic debugging	Top-down testing
		Visualization/animation
Formal	Induction	Inference
	Logical deduction	Inductive assertions
	Lambda calculus	Predicate calculus
	Predicate transformations	Proof of correctness

between the different abstraction levels, each one represented by a specific model.

Several techniques and tools make verification and validation possible in different domains [27,28]. Love and Back [29] propose some common ones for Enterprise Modeling, summarized in Table 1.

These go from the most informal ones more suitable for validation to the most formal ones more dedicated to verification. To summarize this table, validation remains difficult without any human experts and verification remains poorly developed. This article will focus on verification based on formal principles. Indeed, formal methods [30,31] and tools [28] offer the promise of significant improvements in verification.

They are the only way capable of demonstrating the absence of undesirable model behavior and of making sure that the model will function as required. On the other hand, it is widely recognized that these methods are expensive, not easy to use and therefore not very accessible. The reason is that formal approaches cannot be understood and handled by most experts in the domain, and especially because they are not yet used in the EM field. As a matter of fact, their use is not always very interesting for a non-specialist because it lays down too strict and a limited vision of the modeled system and leads to uncertain delays before obtaining the needed results. This paper presents the concepts and tools supporting an Enterprise Modeling Verification methodology based on formal proof of properties, taking into account the following main concepts and mechanisms:

- A modeling language enabling a property to be described and handled [32].
- An ontology allowing us to develop a common vocabulary [33] composed of sets of related concepts and relations dedicated to Enterprise Model Verification. This ontology allows us to propose extensions to UEML.
- A Properties Reference Repository (PRR) [34] is a pre-defined properties database in which the user can choose, select and specify different properties which classically have to be proved on a model. We describe an EM PRR below.
- A set of formal verification mechanisms [35,36] based on Conceptual Graphs [37].

In order to demonstrate this work, the proposed approach will be applied to UEML Version 1.0 [20], whose meta model is presented in Fig. 1.

UEML should be able to translate a business process model into another modeling language without any distortion of

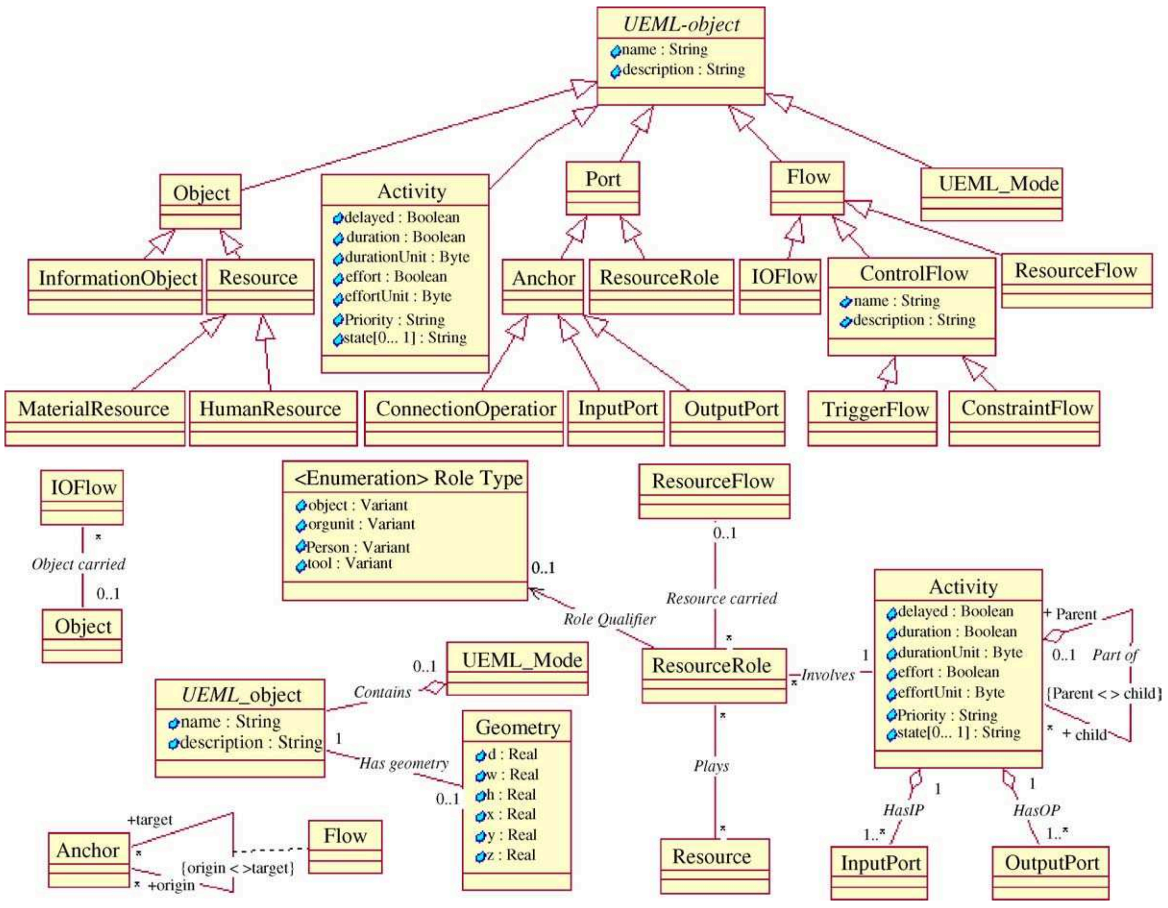

Fig. 1. UEML 1.0 meta model [20].

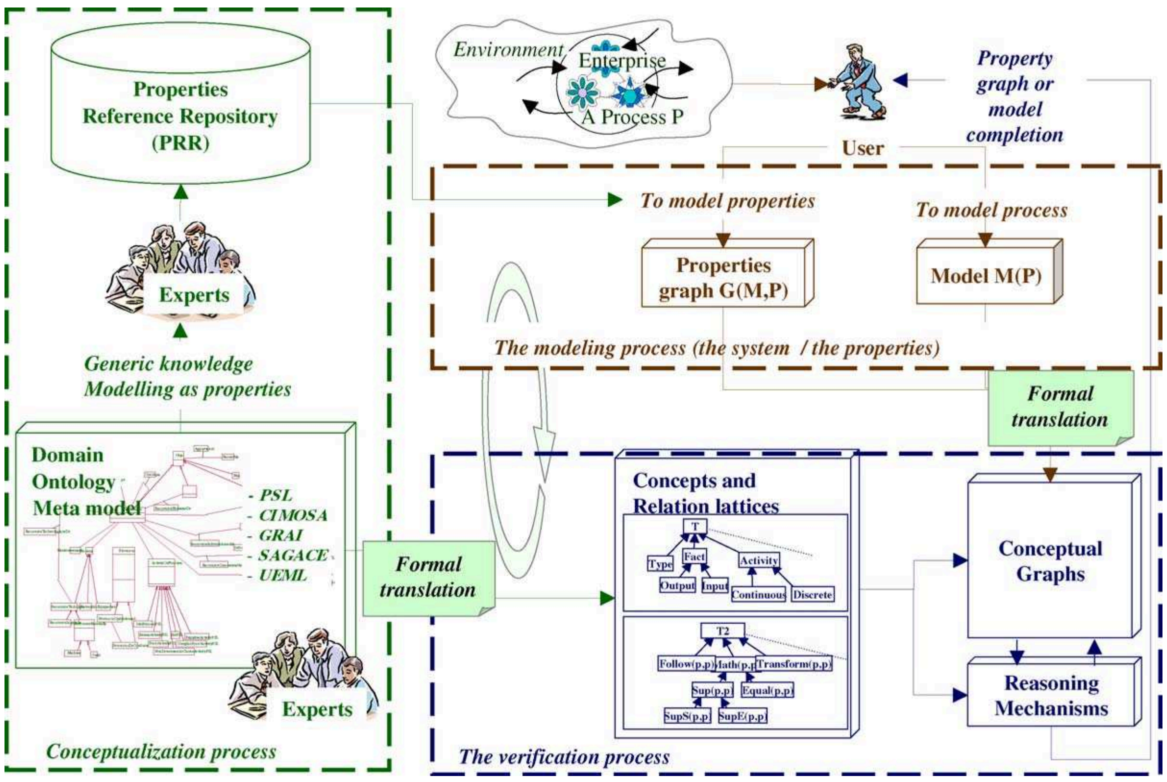

Fig. 2. The methodology is described as three interconnected processes.

semantics. So, it will be considered here as a *modeling language* even though that is not its prime objective. All the proposed concepts, relations and mechanisms can be added as proposed to UEML 1.0 enabling verification tasks to be performed during the modeling process.

2. Principles of the methodology

The proposed methodology may be described by three interconnected processes illustrated in Fig. 2 and detailed in the following sections:

- *Conceptualization process*: Before being able to verify a model, a set of experts defines a formal ontology for the selected domain and builds a domain-oriented Properties Reference Repository.
- *Modeling process*: The user represents a part of an enterprise using UEML and specifies the properties corresponding to the needs he or she wants to verify, using the Property Reference Repository defined in the previous phase.
- *Verification process*: The user must then prove some chosen properties on the model by use of formal mechanisms allowing him to increase his own knowledge and improve the model's quality and relevance. The user in charge of an enterprise model will be involved only during the modeling and verification processes, whereupon the conceptualization process provides a PRR and common ontology available to all the users.

3. The conceptualization process

3.1. From the property concept . . .

As proposed in Ref. [32], any real world entity (a system, a component, an organization) is characterized by a set of *properties*. It is therefore necessary to formalize these properties during the modeling process [9,32] and to analyze them in respect to the real entity. In the same way, any model of a real world entity is itself characterized by a set of properties that also have to be formalized during the modeling process. A property is thus:

- *Knowledge to be modeled*: The modeling language used must enable these properties to be formalized or must include mechanisms and concepts that support this formalization. This knowledge concerns the behavior (evolution rules of the model, evolution laws of the system as proposed in Ref. [38], temporal hypotheses, event occurrence descriptions, auto-adaptive or auto-organization abilities of the entity and so on), the structure and function of the entity or model (functional and non-functional requirements, composition, links and interactions between components and so on). This knowledge may be:
  - The characteristics of real entities (characteristics specific to the entities). They may describe a temporal aspect (activity duration, implementation date, date of birth, etc.), a spatial aspect (position of an actor in a hierarchy, geographic position of the enterprise, speed, acceleration,

etc.) or morphological aspect (dimension, organization, structure, financial costs, etc.).
- The characteristics of the model used (structure, decomposition rules, behavioral evolution rules, etc.).
- Interactions between the entity (the model) and other entities (models) and with the environment if temporal constraints need to be considered. Several new properties can emerge from this interaction.
- Relations existing between other properties characterizing the entity (the model).

• *Knowledge to be handled*: During the modeling process, the user, who is a non-specialist of knowledge modeling, must specify, modify and improve the representation of an entity or model by using properties.
• *Knowledge to be checked*: During the verification process, the user must argue in an indisputable way, as formally as possible, that all the necessary properties have been proven, that is to say they are true. This allows the user to improve his level of knowledge about the behavior and the structure of the entity or model. Some particular mechanisms, such as those presented below, then enable reasoning, proof and, if possible, the identification of emergent properties, i.e. the characteristics of the interrelations between entities or models which were hidden or forgotten during the modeling process.

The CRED model [32,33] allows us to specify and manipulate a property as a *causal relation* between two sets of data called facts.[1] These two sets are named *Cause* and *Effect*, respectively, and the causality relation postulates that there is a law by which the occurrence of the *Effect* depends on the occurrence of the *Cause*. This causal relation is a typed and constrained relation:

• *Logical*: It describes relations of implication and equivalence (reciprocity between cause and effect).
• *Temporal*: It describes temporal links, such as antecedence, in which the cause must be prior to, or at least simultaneous with, the effect.
• *Influence*: *Knowledge about a particular cause modifies the opinion about the verification of the effect* [39]. It thus defines how causes and effects must be linked with respect to particular events or situations. A sense of variation is associated to each influence relation, and can be interpreted as a beneficial or harmful influence on the effect.
• *Emergence*: Each modeled system can be described by some characteristics which are not directly deducible from the characteristics of its components but which result from relations between these components. The explanation of this kind of property needs to take into account all the interactions and feedbacks which connect the entity or model with its environment.

[1] A fact may be an assertion about a parameter, a variable of any type (integer, boolean, real, string, character) contained in the model, that is to say one of the elements of the model, or it may describe information added by the user considering the modeling domain (for example, another property). All facts are gathered in the set F.

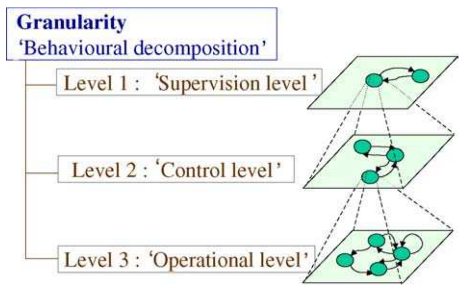
Granularity
'Behavioural decomposition'
Level 1 : 'Supervision level'
Level 2 : 'Control level'
Level 3 : 'Operational level'

Fig. 3. Example of granularity.

This typology allows us to define a property classification that takes into account the following factors:

• *Time*: A static property remains always true or true for a given time interval as proposed in Temporal Logic [40,65], which considers temporal operators, such as 'always' or 'for all *t*'. A dynamic property depends on the evolution of time (wear rate of a tool, financial cost taking into account variation in material costs, etc.).
• *The level of detail* that is employed during the modeling process. A property is defined at a given level of detail by using the information avoidable at this level. Some information may be hidden in sub-levels or upper levels, so a level of *granularity* is defined by the user. This defines the various levels of detail and fixes the ordered relations between these levels. An example of granularity is shown in Fig. 3. In this case the relation may, for example, comply with behavioral decomposition rules of the modeling language.
• *The property's objective*: An "own property" (one for which the set of causes is empty) describes an attribute of an entity irrespective of its environment (color of an object, ability level of an actor). A conjunction property, on the other hand, describes an attribute of the network interaction in which the entity is involved.

This classification allows us to describe, for example, a static own property (the color of an object cannot be modified) or a dynamic own property (the skill level of a human resource evolves). Any combination is possible. The CRED model defines a property as:

$$P = \langle C_P, R_P, E_P, D_P \rangle$$

where

• Causes $C_P = \{\text{fact}|\text{fact} \in F\}$, $\text{card}(C_P) \geq 0$.[2]
• Effects $E_P = \{\text{fact}|\text{fact} \in F\}$, $\text{card}(E_P) > 0$.
• Detail level $D_P = \langle Name,\ G\rangle$ is the level of detail named *Name* of which P is defined in a given granularity G ($D_P \in G$).
• Relation R_P is the relation defining the causal link between Causes and Effects. It is defined by the 4-tuple:

$$R_P = \langle \text{Type}, S_P, \theta_c, \theta_e, \theta_p \rangle$$

where

[2] Set C is empty for each property of type 'own property'.

- *Type* denotes the relation type (logical, temporal, influence, emergence).
- $S_P = \{\text{fact}|\text{fact} \in C_P \cap E_P\}$ is the set of common facts (parameters and variables) which are considered simultaneously as a cause and an effect of P (for example, a variable named t which denotes time).
- The Boolean function θ_c describes under which conditions (by interpretation of causes), the property is verified. θ_c is defined as follows:
  θ_c: $C_P \cup S_P \to \{\text{true, false}\}$; If there is an empty cause then $\theta_c = \text{true}$.
- The Boolean function θ_e describes what are the results on the effects (by interpretation of updating functions) when the property is verified. θ_e is defined as follows:
  θ_e: $E_P \cup S_P \to \{\text{true, false}\}$.
- The Boolean function θ_p describes the constraint under which the property P is verified. θ_p is defined as follows:
  θ_p: $S_P \to \{\text{true, false}\}$.

3.2. *Example*

A manufacturing process describes the transformation of a product. Fig. 4 shows a part of this process, made up of two activities named 'Control conformity' and 'Stock'. Each activity is seen here as a processor which modifies the shape, space and/or time attributes [41] of one or more of its inputs in order to provide one or more outputs, taking into account objectives, constraints, rules and resources. There are four types of activity:

- *Transformation*: A transformation activity implies that all the input flows (bill of materials, information and data, financial and so on) are modified by the activity into output flows and that all the resources used are modified (material wear of a machine, for example). This modification changes one or more attributes of every input and every output flow.

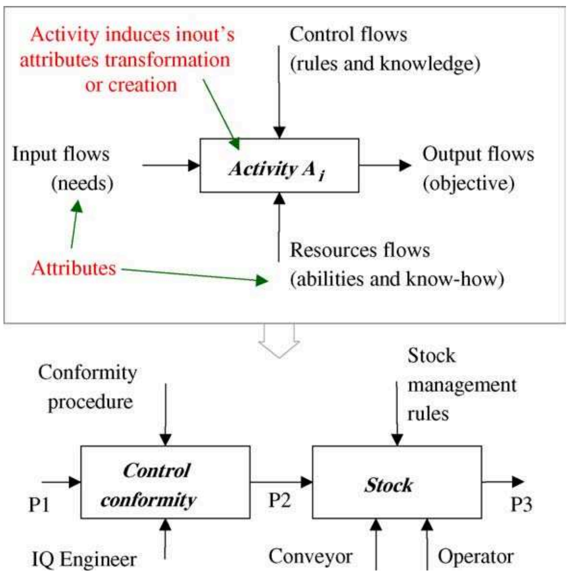

Fig. 4. Short example.

- *Decision*: A decision activity needs to perform and argue for a decision. The resources supporting this activity must be able to take the decision by considering and trusting the information contained in the input flows coming from other activities or from the environment.
- *Control*: A control activity consists in analyzing all the attributes of an input flow in order to verify whether they respect the needs specified by the control input flows.
- *Measure*: A measure activity consists in obtaining (a set of) information about a given (set of) object(s). These can be a product, a piece of information, a set of resources and so on. The aim of this activity is therefore to evaluate, estimate, compute or obtain physical information about this (set of) object(s) and to create an output information flow that enables something to be controlled or decided concerning this (set of) object(s).

In our example, 'Control conformity' is a Control activity type. The property P indicates that each control activity (one or several control operations on a flow value whatever is the type of this flow (information, resource, material, etc.) are executed during the activity), must be followed by a decision activity that takes into account the result of this control operations results. In other word, P can be written as:

$$\{[\forall A|A \in \text{Activities}]\}, \{[A.\text{Type} =' \text{Control}']$$
$$\Rightarrow [\exists B|B \in \text{Activities}, (A <> B) \wedge$$
$$\times (\text{B.InputFlows} \subseteq \text{A.OutputFlows}) \wedge (\text{B.Type} =' \text{Decide}')]\}$$

Then, the elements of P are:

- $C_P = \textit{Activities}$ where *Activities* is the set of the activities of the process.
- $E_P = \textit{Activities}$.
- Relation R_P is of type Logical Implication ($\Rightarrow$) with:
  - $S_P = \textit{Activities}$.
  - $\theta_p = [\forall A]$ which select all instances of activity.
  - $\theta_c(C_P \cup S_P) = [\text{A.Type} = \text{'control'}]$ which returns true if the instance A of activity is of type 'control'.
  - $\theta_e(E_P \cup S_P) = [\exists B|B \in \textit{Activities}, \quad (A<>B) \quad \wedge(\text{B.InputFlows} \subseteq \text{A.OutputFlows}) \wedge (\text{B.Type} = \text{'Decide'})]\}$ which returns true for each instance B of an activity which is different from A, where the outputs of A are contained in the inputs of B, and B is of the type 'decide'.
  - D_P is the detail level of the activity model.

In this case, it is impossible to prove the property P: perhaps the type of the activity called 'Stock' does not match. This may highlight a modeling error or other mistake, enabling the user to rectify the model.

The CRED model allows us to describe the properties of an entity. All the properties are represented in the same manner, thus guaranteeing the homogeneity of the representation. However, the specification of a property remains difficult. The following section presents the concepts we propose to help the user during the modeling process.

3.3. . . . To the Property Reference Repository . . .

The user may ask the following types of questions:

- What is (are) the required property(ies) which have to be specified in order to be able to verify and, if possible, validate a part of my model?
- Is there a generic wording of some classical properties available for my problem, even if it is proposed at a higher level of abstraction?

Our approach proposes the definition of a database, named the Properties Reference Repository, that lists and maps fundamental and generic properties to help, structure and reduce the user's efforts during the modeling process. No property can be proved directly on a model without interpretation. The following property typology helps the user to select the relevant properties from the PRR:

- *Axiomatic properties*: An axiomatic property is considered as an axiom of indubitable knowledge. Such properties primarily characterize the environment or intrinsic characteristics of a system or the modeling language used, which are considered as always verified from the user's perspective. They are inspired by taking into account theoretic system engineering, modeling standards and if necessary the user's experience. For example, they may translate the laws of nature in order to explain certain phenomena external to the studied model (such as Maxwell's equations of electromagnetism or Newton's laws of gravity).
- *System properties*: According to a system engineering modeling approach [42], a system property is '*perceptible or given or fixed by an observer to represent the purpose of a system, i.e. the objective fixed for the system, without taking into account possible alterations to its environment, at least over the given period*'. A system property describes:
  - *A system (set of) constraint(s)*: deployment, geographical, architectural, functional, safety, confidentiality, maintainability, environmental, volume, performance, availability, accessibility, usability or usage, etc. These properties describe constraints which have to be respected, that is to say the requirements and expectation of the user.
  - *A system (set of) characteristic(s)*: "own" characteristics (such as color, speed, dimension, etc.), performance, mission, objective, composition, interactions between component properties, input/output relations, current behavior, etc.
  - *A negative characteristic or constraint*, which cannot be verified by the model of the system but have to be verified by the model of the anti-system. The anti-system is a system that is the exact opposite of the studied system. These properties enable the user to specify what the system must not do or not be able to do, the characteristics it must not have, etc.
- *Meta model and modeling language properties*: These properties:
  - Characterize the capabilities of the model, that is to say its structure, behavioral semantics (impact of the behavioral rules on the model) and temporal evolution.
  - 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.

This typology enables us to define either a PRR suitable for specific domain of application and to describe the design process of a PRR. It consists of the three tasks detailed below, managed by a group of experts:

- *Experts' Task 1*: *To build a vocabulary*. A model, essentially for reasons of ease of representation and communication, readability, relevance and formal use, is always built up on the basis of a limited and fixed list of common concepts, relations between those concepts, hypotheses and rules (for

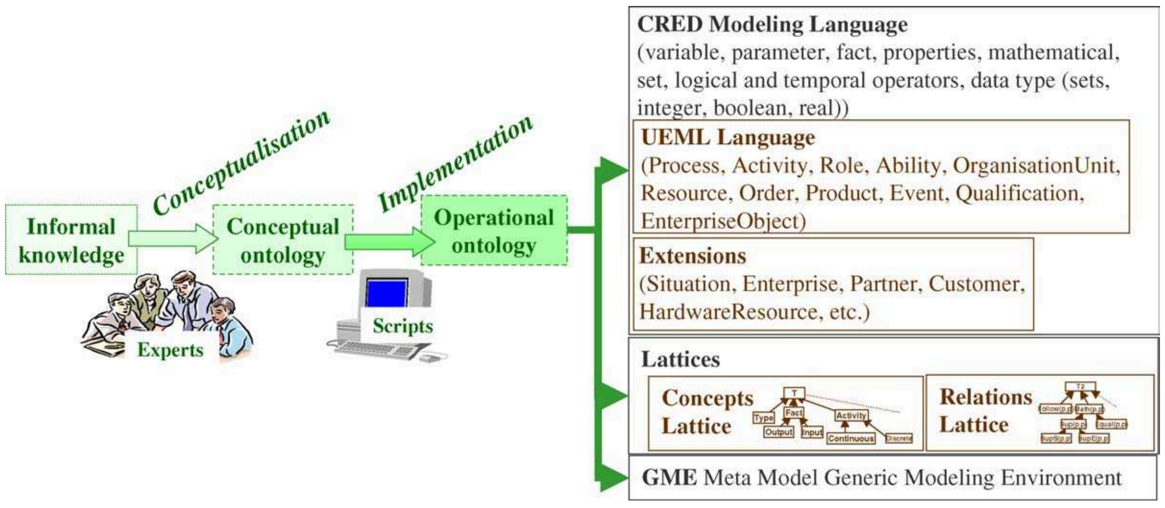
Conceptualisation
Implementation
Informal knowledge
Conceptual ontology
Operational ontology
Experts
Scripts
CRED Modeling Language
(variable, parameter, fact, properties, mathematical, set, logical and temporal operators, data type (sets, integer, boolean, real))
UEML Language
(Process, Activity, Role, Ability, OrganisationUnit, Resource, Order, Product, Event, Qualification, EnterpriseObject)
Extensions
(Situation, Enterprise, Partner, Customer, HardwareResource, etc.)
Lattices
Concepts Lattice
Relations Lattice
GME Meta Model Generic Modeling Environment

Fig. 5. Conceptualization and implementation of the ontology.

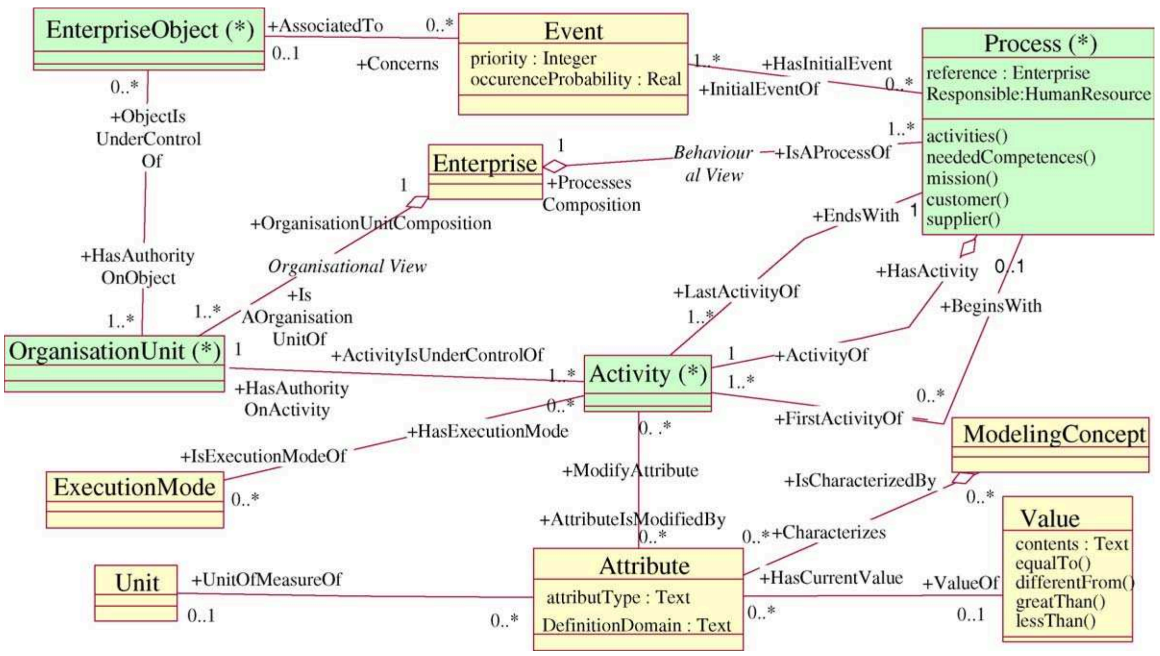

Fig. 6. Partial view of the proposed ontology meta-model.

example, cardinality constraints on each relation). Verification requires the completion and enrichment of the base vocabulary of UEML, by defining relevant concepts and associated relations using the property model proposed in CRED. An ontology is therefore proposed in order to describe these concepts and relations. An ontology is a '*conceptualization of the real world*' [43] and, in our case, the ontology needs to include concepts and relations considered as commonly accepted by several methods in the Enterprise Modeling domain and already defined in the UEML language (see Fig. 1). Other existing ontologies, such as TOVE and PSL and some related works on enterprise system engineering [41,42,44,45] have been merged, using the conceptualization and implementation approach summarized in Fig. 5. It is then necessary for the user to have at his/her disposal the extensions required for manipulating the properties in compliance with the CRED model. A part of the main result is presented in Figs. 6 and 7 in the form of a meta-model of the proposed ontology.

For more clarity, this meta-model was built using the Unified Modeling Language (UML) class diagram format [46] which is now often used for meta-modeling. UML offers a good compromise between power of description (a relation is constrained by cardinality and role, possibility of progressive enrichment of concepts using inheritance links, etc.), readability, existing works and reference in the domain [47] and a sufficient formalization level. The meta-model was implemented using Rational Rose®. Object Classes proposed without any ambiguities by UEML 1.0 [20] or the UEML meta model proposed by [21] are marked by a (*). All other meta-entities have been added in order to complete the ontology to address the property verification objective. Using scripts, Rational Rose® allows documentation to be generated in order to help the experts to validate their meta-model as proposed in the NIAM-ORM approach [48] throughout the ontology conceptualization process, and to implement the obtained ontology that is to say to extract and to translate the object classes and relations that make up the vocabulary. Each object class, attribute, method and relation of this meta model has been checked and the resulting modeling language may be considered as an extension of UEML, interoperable with UEML 1.0. This new language is now usable using the modeling environment GME [49]. This tool allows the user to describe his/her own models of processes as shown, for example, in Fig. 8, which highlights some accepted modeling concepts and relations of the new language.

Finally, the hierarchy of concepts described in the meta-model is translated into a 'Concepts lattice'. At the same time, all relations are gathered into a 'Relations lattice' respecting the rules given in Fig. 9. These two lattices will be necessary for the last phase of the verification process.

- *Experts*' *Task 2*: *To describe generic properties*. It seems desirable to guide or to help experts during this stage, particularly for generating Model properties and System Properties. The approach encourages them to settle the following questions[3]:
  - *What does a concept currently do and/or what should it do*? This is a question of representing the properties that characterize the functional aspect attached to every concept.

[3] The questions must be answered by considering their positive form—the properties focus on the system and the model or their negative form—the properties focus on the anti system. This will reveal any psychological inertia on the part of the experts during the design task.

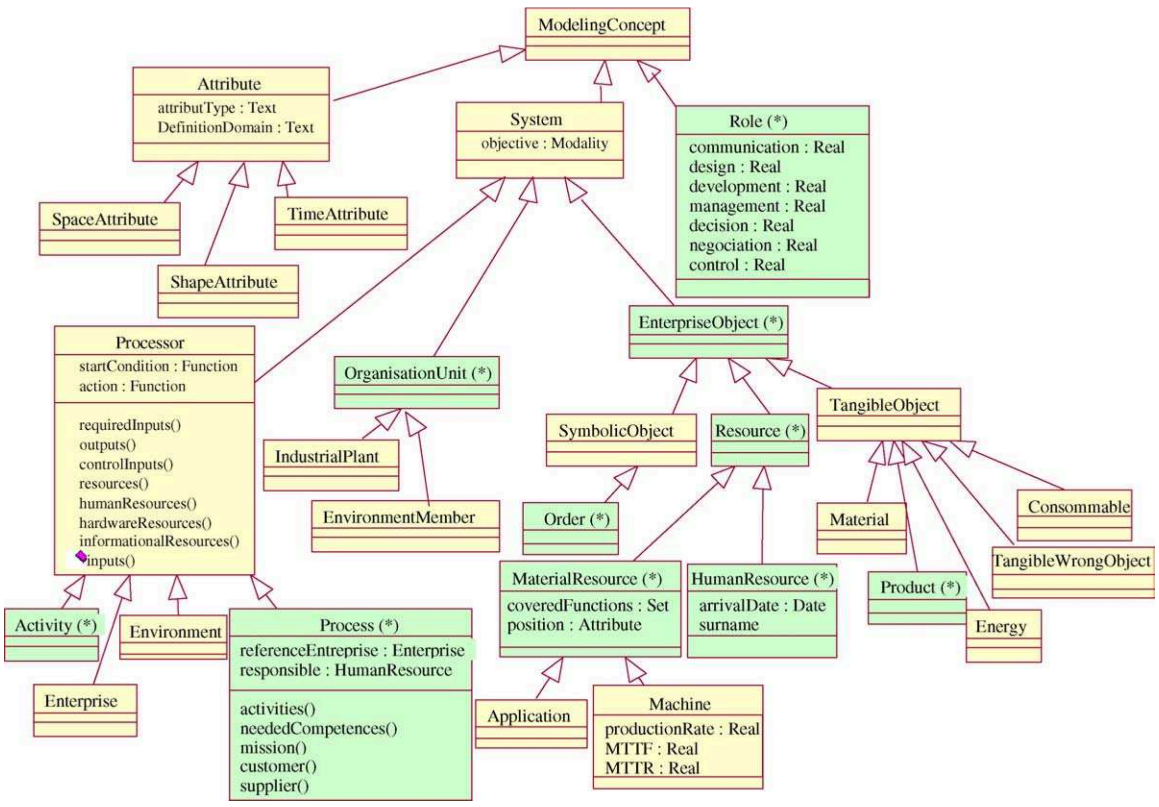

Fig. 7. Partial view of the inheritance tree used to describe the ontology.

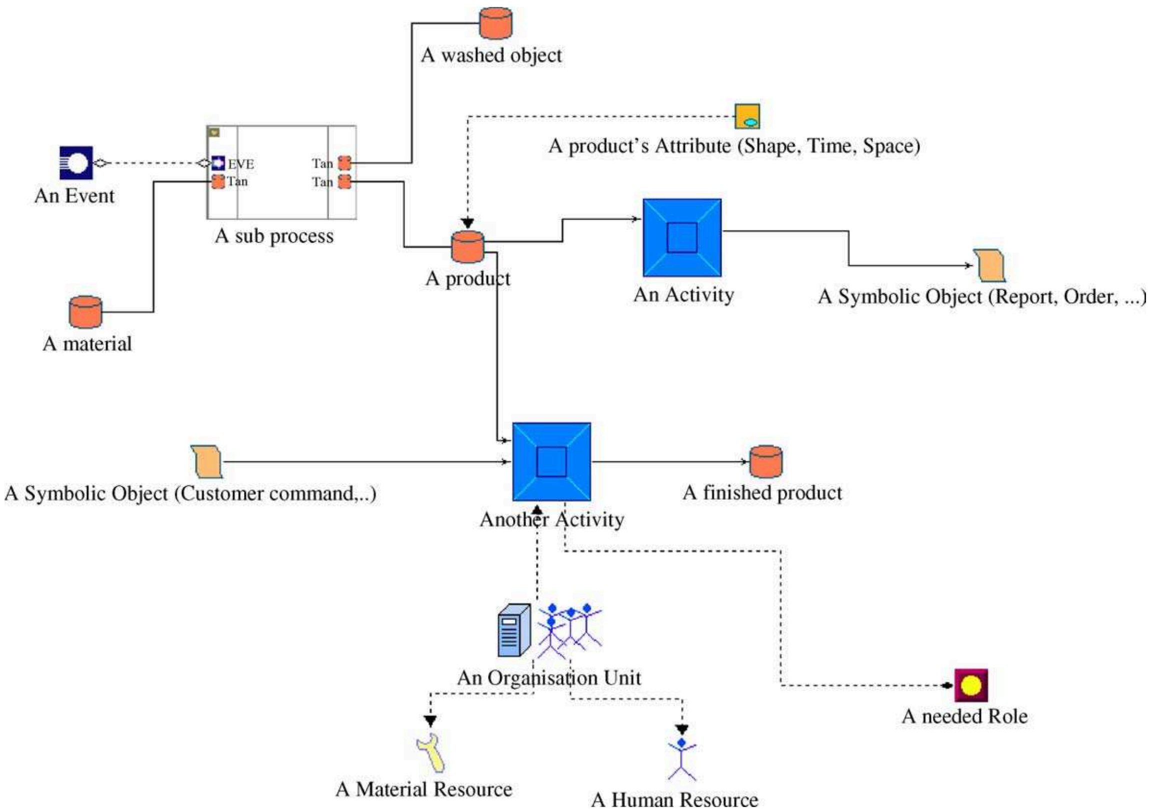

Fig. 8. Example of a process model using UEML 1.0 and GME [49].

Object Diagram (UML)		Lattice
Class	↔	Concept
Inheritance	↔	Concept hierarchy
Encapsulation	↔	Nested concepts
Method	↔	Relation
Relation	↔	Relation
Attribute	↔	Relation

Fig. 9. Translation rules used for lattice construction.

For instance, a system must have a purpose; any activity must be characterized by a performance objective and so on.

- *How does it evolve and/or how should it evolve in time (for every concept)*? This is a question of representing the properties that characterize the behavior of each concept. For instance, in a process that is initiated by an event, the firing rules of a transition in the model (for example, if the model is a Petri net) cause the evolution of the marking vector by respecting a hypothesis of synchronism and parallelism.
- *What is the composition of each concept and how is it structured through these components, themselves other concepts*? The answer enables the properties characterizing the structure of each concept to be identified. For instance, a network of activities in the GRAI approach is composed of one or more decision-making activities and of zero or more execution activities; the modeling of an assembling activity must necessarily show two or more input flows of objects and one or more output flows of objects.
- *Of what information, knowledge or data is every concept the source, the destination or the container*?
- *With which other concept(s) does a concept interact*?

• *Experts*' *Task 3*: *To classify the properties obtained* by establishing the coherence and completeness of the Repository. In the present state of study, this stage is assured through critical examination by the experts. Mechanisms to analyze dependency and coherence are explored in our studies. Fig. 10 shows a part of the PRR obtained, considering the UEML case.

System Properties	Every activity defined for each of its human resources includes a list of responsibilities and missions to be filled
	Every activity can require technical means (hardware or software) requiring capable human resources to use it (these persons are considered to be human resources of this activity)
	Any material means (implementing attribute transformation in the broad sense) has characteristics particularizing its capability (maximum capability of stocking, competence, diameter of drill minimum and maximum, speed of rotation, speed, ...).
	An activity transforms (independently or at the same time) one or several attributes of time, space and form of objects carried by input flows to give attributes of time, space and shape of one or some of the objects carried by output flows
	One or several output flows of an assembling activity must be a (set of) tangible object(s)
	One or several output flows of data manipulation activity must be constituted by intangible objects acquired by aggregation, fusion or copy of one or some of the input flows
	A resource (material or human) involved in an activity must be available and able to fill its task or the operation which is assigned to it
Model properties	If a case of non determinist activity happpens at an instant t and if this activity is broken down into sub-activities then there exists an occurrence at t of one or some of these sub-activities possibly at different instants
	One resource must be in a non prduction state, a waiting state, a production state or a led stopping state.
	For each state, only one output transition of this state can be validated at each instant (exclusivity of the model evolution)
Axiomatic properties	An atomic activity AO has a length of occurence i (named AOi) in which BeginOf (AOi) = EndOf (AOi)
	Duration(activity) =endof (activity) - beginof (activity)

Fig. 10. Partial view of properties reference repository (PRR).

4. The modeling process

Taking into account the model M of a given part of enterprise to be analyzed, the user has to specify the properties by carrying out the following tasks:

- *User's Task 1*: *To identify* and *select* a set of relevant or interesting properties in the Properties Reference Repository depending on the user's point of view and modeling objectives (behavioral analysis of a system, performance analysis, etc.).
- *User's Task 2*: *To formalize* this set of chosen properties. Each property is specified in the PRR at a high level of abstraction. It has to be formalized taking into account the facts (extracted from the model M or specified by the user) in order to give it some meaning, by means of the CRED modeling language and considering the vocabulary which has been proposed by the experts. This task is supported by a Unified Property Specification Language (UPSL) [50] tool which implements the CRED model.

Due to the dependence between some of the specified properties (e.g. the same fact has been used in different properties which become dependent on one another), the result of the properties specification phase is a non-connected graph. This gathers all the knowledge the user has added to the model. All the properties describing this knowledge now have to be verified.

5. The verification (and, as far as possible, validation) process

Indeed, CRED allows us to formalize knowledge and it is then sometimes possible and interesting to translate this knowledge into other formal languages (temporal logic, CTL, TCTL, Z or others) as input for theorem provers, such as Z-Eves [51] or model checkers, such as PVS [52], STEP or SMV [53,27].

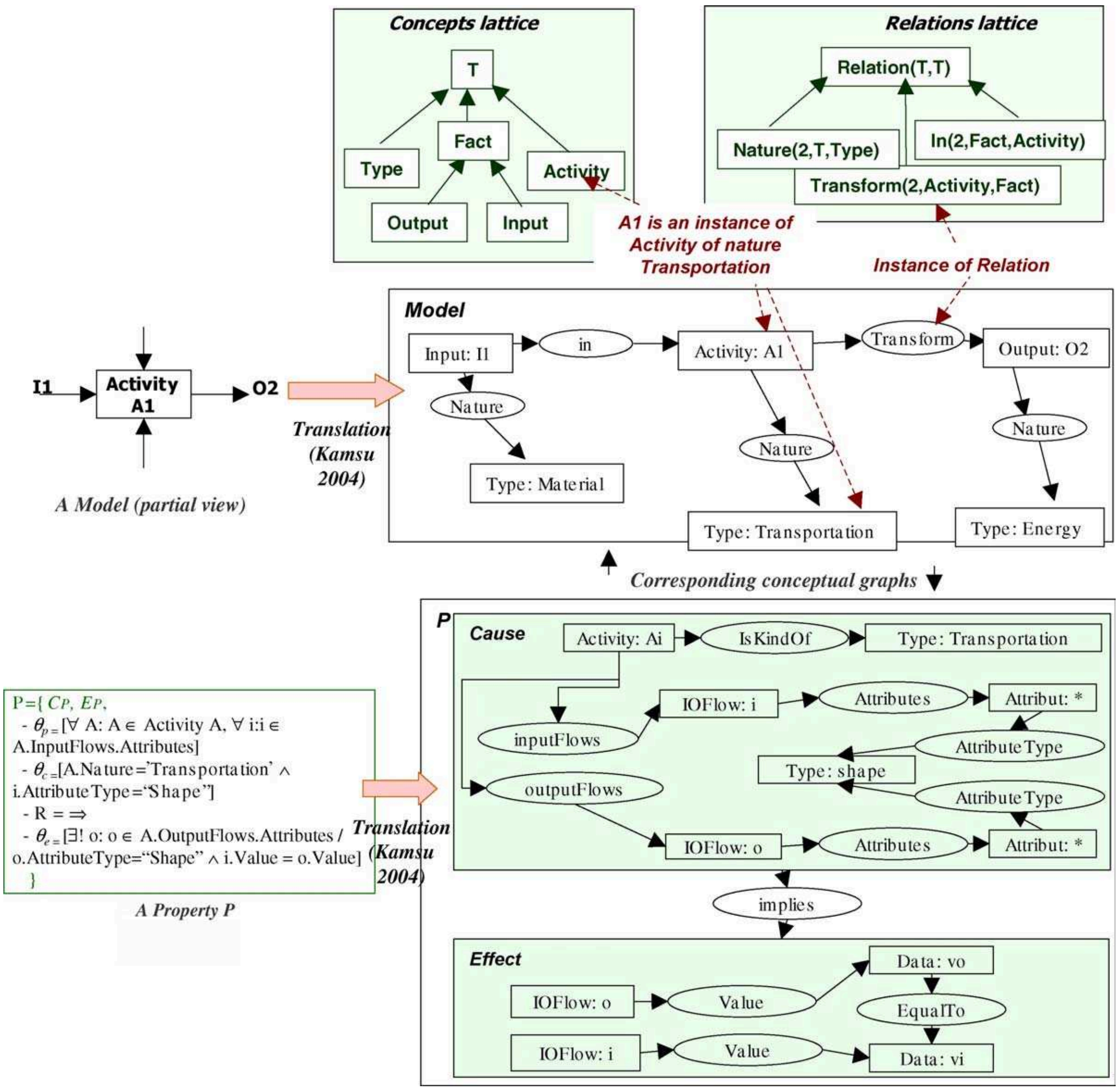

Fig. 11. Verification using conceptual graphs [37].

A relatively recent trend in formal techniques, often called "lightweight formal methods" [31], has shown potential for detecting major errors in requirements statements, without the expense of formal design verification [54], by applying formal analysis to earlier products and models of the system design process.

In particular, conceptual graphs [37] are a language of knowledge representation [55] that enables the user at the same time to define a vocabulary (i.e. an ontology that closely corresponds to the concepts/relations relevant to Enterprise Modeling) and to use this vocabulary to conceptualize facts. A simple conceptual graph is a finite, connected, directed, and bipartite graph composed of two kinds of nodes called *concepts* and *conceptual relations*. Conceptual Graphs can be considered as a compromise between formal language and graphical language because they are visual and incorporate a range of reasoning processes. The graph operations (projection, rules and constraints [56]) provide several means of analysis, for both quantitative and functional properties (completion time, workloads, critical path, data flow, process type and so on). While the formality of graph operations does not provide any guarantees, it helps to increase confidence in the model by demonstrating that some classes of error are not present.

The methodology aims to translate the model to be analyzed into a global conceptual graph highlighting concepts and relations between concepts which are described in the model. Each property that has been specified by the user is translated from a formal manner into another conceptual graph. Each translation (model or property) is made by using an appropriate algorithm [33]. Finally, projection mechanisms, rules or a constraints scheme may be used in order to prove the chosen property that is to say to compare the conceptual graph describing the property to the conceptual graph describing the model.

Fig. 11 illustrates how a given model (in this case, an activity is only represented at a detail level named 'Operation') is translated into a conceptual graph and the lattices of concepts and relations defined in the previous section by the experts. The user has chosen the following property: '*If the activity aims to transport a product from one point to another one without any transformation, the outputs of the activity must have the same shape attributes (no transformation of the value) as its inputs (none of the shape attributes of an input should be modified by the activity)*'. This property P is then described in CRED with:

- C_P = Activities.
- R_P = {Activities, [$\forall A|A \in$ Activities, $\forall i|i \in$ A.InputFlows. Attributes],[A.Nature = 'Transportation' $\wedge$ i.AttributeTyp-i.AttributeType = "Shape"], $\Rightarrow$, [$\exists!$ o: o $\in$ A.OutputFlows. Attributes/o.AttributeType = "Sha-pe" $\wedge$ i.Value = o.Value]}.
- E_P = Activities.
- D_P = 'Operation'.

Let G be the graph representing the model, P be the query graph representing this property, R be a set of implicit knowledge rules and C be a set of constraints depending on 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. If not, Conceptual Graph theory offers some means to establish the possible causes for the non-verification of P. This enables defaults or mistakes to be highlighted and the model to be improved.

6. Conclusion

This paper proposes a methodology for achieving the verification, and to some extent the validation (V&V) for Enterprise Modeling. This methodology has been applied here to a common and relevant modeling language named UEML and is supported by a set of interconnected tools:

- The Unified Property Specification Language [50] framework to provide the property specification and handling mechanisms.
- Rational Rose® for building parts of the Ontology and generating interchange files (XML, CogXML and SQL) so as to establish communication without loss of meaning between all the other tools.
- Generic Modeling Environment (GME) [49] used for UEML processing and model translation.
- Cogitant 5.1.4 [57] to provide verification mechanisms.

This approach has multiple interests. The requirements, characteristics and, constraints which cannot be described using UEML may be specified by means of a unique knowledge modeling language called CRED, which then enables Conceptual graphs, a theorem prover or a model checker to be used for the verification.

There are numerous perspectives for future work. First, Conceptual Graphs cannot describe temporal evolution in a simple way. Further research will develop this aspect. Second, this kind of analysis approach may enable a simulation process to be guided by helping the user to choose particular scenarios. In the same way, the impact on the system's behavior on possible improvements and ameliorations that are classically used in industry (such as proposed by [38,58,59]) may then be described and tested in order to modify the model to obtain a 'TO BE' model. This will be done in order to help the user to select a potential solution and to improve the performance of a business process.

Last but not least, a study is ongoing and appropriate tools are under construction in the risk analysis domain [60] in order to improve the risk management toolbox [61,62].

References

[1] F.B. Vernadat, Enterprise Modeling and Integration: Principles and Applications, Chapman & Hall, 1996.

[2] A. Molina, H. Panetto, D. Chen, F. Vernadat, L. Whitman, Enterprise Integration and Networking: Milestone Report, TC 5.3 Enterprise Integration and Networking to Appear in ICEIMT2004, International Conference on Enterprise Integration and Modeling Technology, Canada, 2004

[3] P. Bernus, K. Mertins, G. Schmidt, Handbook on Architectures of Information Systems, Springer, 2003.

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

[5] C.P. Menzel, R.J. Mayer, The IDEF family of languages, in: P. Bernus, K.et Mertins, G. Schmidt (Eds.), Handbook on Architectures of Information Systems, Springer, Berlin, 1998.

[6] AMICE Consortium AMICE, CIMOSA: Open Architecture for CIM, Berlin, Springer-Verlag, 1993.

[7] G. Doumeingts, B. Vallespir, D. Chen, GRAI grid decisional modeling, in: P. Bernus, K. Mertins, G. Schmidt (Eds.), Handbook on Architectures of Information Systems, International Handbooks on Information Systems, Springer, 1998.

[8] NIST Process Specification Language, see http://ats.nist.gov/psl/, 2002.

[9] PSL Property Specification Language Reference Manual, Accelera Formal Verification Technical Committee (FVTC), Version 1.1, see http://www.eda.org/vfv/, 2004.

[10] M.S. Fox, The TOVE project: a common-sense model of the enterprise, industrial and engineering applications of artificial intelligence and expert systems, in: F. Belli, F.J. Radermacher (Eds.), Lecture Notes in Artificial Intelligence #604, Springer-Verlag, Berlin, 1992, pp. 25–34.

[11] A. Arkin, Business Process Modeling Language BPML 1.0, see www.BPMI.org, 2002.

[12] A.-W. Scheer, ARIS, Business Process Frameworks, second ed., Springer-Verlag, 1998.

[13] T.J. Williams, G.A. Rathwell, H. Li (Eds.), PERA A Handbook on Master Planning and Implementation for Enterprise Intergration Programs Based on the Purdue Enterprise Reference Architecture (PERA) and the Purdue Methodology, Purdue Laboratory for Applied Industrial Control, 2001.

[14] GERAM GERAM: Generalised Enterprise Reference Architecture and Methodology Version 1.6.1, IFIP–IFAC Task Force on Architectures for Enterprise Integration, 1999.

[15] D. Chen, B. Vallespir, G. Doumeingts, Designing Manufacturing Systems: Contribution to the Development of an Enterprise Engineering Methodology (EEM) within the Frame of GERAM, IFAC'2002 World Congress, Barcelona, Spain, 2002.

[16] ISA Enterprise—Control system Integration, Instrument Society of America, Part 1, ISA-ds95.01, Draft 14, 1999.

[17] ITU Information Technology, Open Distributed Processing, Reference model, Enterprise Language, Telecommunication Standardization Sector of International Telecommunication Union, Recommendation X.911, Draft 7, 2000.

[18] D. Chen, Vernadat Enterprise Interoperability: A Standardization View, Handbook on Architectures of Information Systems, Springer, 2003.

[19] M. Petit, G. Doumeingts, Enterprise Modeling State of the Art, Deliverable D1.1 of the UEML Project, November 2002, see www.ueml.org, 2002.

[20] UEML Deliverable D3.1: Requirements Analysis: Initial Core Constructs and Architecture, Unified Enterprise Modeling Language UEML Thematic Network—IST-2001-34229, see www.ueml.org, 2003.

[21] F.B. Vernadat, UEML: towards a unified enterprise modeling language, in: Proc. 3ème Conférence Francophone de Modélisation et Simulation (MOSI-M'01), Troyes, France, 2001.

[22] C. Chen, B. Vallespir, G. Doumeingts, Developing an unified enterprise modeling language (UEML)—roadmap and requirements, in: Third IFIP Working Conference on Infrastructures for Virtual Enterprise, Collaborative Business Ecosystems and Virtual Enterprises, Kluwer Academic Publishers, Sesimbra, Portugal, 2002ISBN: 1-4020-7020-9.

[23] Popkin Enterprise Modeling: Aligning Business and Information Technology, White Paper, Popkin Software, see http://www.popkin.com/customers/customer_service_center/downloads/whitepaper/index.htm, 2003.

[24] ISO 8402: Quality Management and Quality Assurance—Vocabulary, second ed., 1994-04-01, International Standard Organization, 1994.

[25] NASA Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Volume II: A Practitioner's Companion, http://www.eis.jpl.nasa.gov/quality/Formal_Methods/document/NASAgb2.pdf, 1998.

[26] J. Schekkerman, Enterprise Architecture Validation, Achieving Business-Aligned and Validated Enterprise Architectures Institute For Enterprise Architecture Developments Report, 2003.

[27] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, P. McKenzie, Systems and Software Verification: Model Checking Techniques and Tools, Springer, 2001.

[28] Yahoda web site presenting an overview of formal verification tools, see http://anna.fi.muni.cz/yahoda/, 2003.

[29] G. Love, G. Back, Model Verification and Validation for Rapidly Developed Simulation Models: Balancing Cost and Theory, White Paper of the Project Performance Corporation, http://www.ppc.com/, 2000.

[30] A. Van Lamsweerde, in: A. Finkelstein (Ed.), Formal Specification: A Roadmap, The Future of Software Engineering, ACM Press, 2002.

[31] D. Jackson, Lightweight formal methods, in: Proceedings of International Symposium of Formal Methods Europe, Berlin, Germany, March 12–16, 2001.

[32] E. Lamine, éfinition d'un modèle de propriété et proposition d'un langage de spécification associé: LUSP, Ph.D. Thesis, Montpellier II University, 2001 (in French).

[33] B. Kamsu-Foguem, Modélisation et Vérification des propriétés de systèmes complexes: application aux processus d'entreprise, July 2004, Ph.D. Thesis University Montpellier II, 2004 (in French).

[34] V. Chapurlat, B. Kamsu-Foguem, F. Prunet, A property relevance model and associated tools for system life-cycle management, in: 15th IFAC World Congress on Automation Control (B'02), Barcelona, Spain, 2002.

[35] B. Kamsu-Foguem, V. Chapurlat, F. Prunet, Enterprise model verification: a graph-based approach, in: Proceedings of CESA'2003, Computing Engineering in Systems Applications, Lille, France, 2003.

[36] V. Chapurlat, B. Kamsu Foguem, F. Prunet, Enterprise model verification and validation: an approach, Annual Review in Control, IFAC Journal (2003).

[37] J.F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, New York (U.S.A.), 1984.

[38] D. Mann, Hands on Systematic Innovation, CREAX Press Editor, 2002.

[39] J. Pearl, Causality: Models, Reasoning and Inference, Cambridge University Press, 2000.

[40] J.F. Allen, Towards a general theory of action and time, Artificial Intelligence 23 (1984) 123–154.

[41] F. Mayer, Contribution au Génie Productique: Application à l'ingénierie pédagogique en Atelier Inter établissements de Productique Lorrain, Thèse de doctorat en production automatisée, Université Henri Poincaré/Nancy I, 1995 [in French].

[42] SAGACE Méthode SAGACE: le systémographe, CEA, Version 1.0, 1999 (in French).

[43] M. Uschold, M. Gruninger, Ontologies: principles, methods and applications, Knowledge Engineering Review 11 (2) (1996) 93–136.

[44] C. Feliot, Modélisation de systèmes complexes: intégration et formalisation de modèles, Ph.D. Thesis, University Lille I, 1997 (in French).

[45] P. Lamboley, Proposition d'une méthode formelle d'automatisation de systèmes de production à l'aide de la méthode B, Ph.D. Thesis University Henri Poincaré Nancy I, 2001 (in French).

[46] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modelling Language User Guide, Addison-Wesley, 1999.

[47] OMG Object Management Group (OMG) web site, see UML, MDA and other related works on http://www.omg.org/, 2004.

[48] G.M. Nijssen, T. Halpin, Conceptual Schema and Relational Database Design, Prentice Hall, Sydney, 1989.

[49] GME Generic Modeling Environment (GME) Version 4 User's Manual, Release 4-2-3, Institute for Software Integrated Systems (ISIS) Vanderbilt University, 2004.

[50] V. Chapurlat, T. Lambolais, F. Benaben, C. Antoine, Unified Properties Specification Language: A Framework in Preprints of INCOM'04 congress, 2004a.

[51] M. Saaltink, The Z/EVES 2.0 User's Guide, TR-99-5493-06a, see http://www.ora.on.ca/z-eves/, 1999.

[52] S. Owre, N. Shankar, J.M. Rushby, D.W.J. Stringer-Calvert, PVS Language Reference, Version 2.4, Computer Sciences Laboratory, SRI International, see http://pvs.csl.sri.com/, 2001.

[53] K.L. McMilan, The SMV System for SMV Version 2.5.4: SMV Manual, see http://www-2.cs.cmu.edu/~modelcheck/smv.html, 2000.

[54] S.R. Rakitin, Software Verification and Validation for Practitioners and Managers, Artech House Publishers, 2001.
[55] J.F. Sowa, J.A. Zachman, Extending and formalising the framework for information systems architecture, IBM Systems Journal 31 (3) (1992) 590–616.
[56] J.F. Baget, Représenter des connaissances et raisonner avec des hypergraphes: de la projection à la dérivation sous contraintes, Ph.D. Thesis, University of Montpellier 2, 2001 (in French).
[57] D.N. Chrorafas, Enterprise Architecture and New Generation Information Systems, Information Technology Coll, St. Lucie Press, 2002.
[58] D. Genest, CoGITaNT Version-5.1—Manuel de référence, see http://cogitant.sourceforge.net, 2003 (in French).
[59] J.B Revelle, Manufacturing handbook of best practices: an innovation, productivity and quality focus, in: J.B. Revelle (Ed.), APICS Series on Resource Management, St. Lucie Press, 2002.
[60] V. Chapurlat, J. Montmain, A. Djamel Gharbit, Proposition for risk analysis in manufacturing and enterprise modeling, In: P. Bernus, M. Fox, J.B.M Goossenaerts (Eds.), Knowledge Sharing in the Integrated Enterprise, Proceedings of ICEIMT'04, International Conference on Enterprise Integration Modelling and Technology, 9–11 October 2004, Toronto, ISBN:0-387-26608-9.
[61] J. Tixier, G. Dusserre, Review of 62 risk analysis methodologies of industrial plants, Journal of Loss Prevention in the Process Industries (2000).
[62] CAS Overview on Enterprise Risk Management, Casualty Actuarial Society ed., 2003.
[63] D. Chen, F. Vernadat, Standards on enterprise integration and engineering—a state of the art, International Journal of Computer Integrated Manufacturing (IJCIM) 17 (April–May (3)) (2004) 235–253.
[64] EN/DIS Language Construct for Enterprise Modeling, EN/DIS 19440, CEN/TC 310/WG 1 Systems Architecture, 2002.
[65] Z. Manna, P. Pnuelli, The Temporal Logic of Reactive and Concurrent Systems, Springer-Verlag, Berlin, 1992.

Vincent Chapurlat has a master's in control command of production systems (1991) and a PhD in control command system specification and verification (1994) from the University of Montpellier II. He is currently assistant professor at the Laboratory of Informatics and Production Systems Engineering (LGI2P) from the Ecole des Mines d'Alès. His research aims to develop and to formalize concepts and tools allowing to help complex systems' designer teams to verify and to validate their design models. The concerned application domains are the Enterprise Modeling and the System Engineering (SE) domains. He is member of the Technical Committee 5.3 'Enterprise Networking' from IFAC Board and head of the 'Verification, Validation and Accreditation of Enterprise Models' sub group.

Bernard Kamsu Foguem has a bachelor's degree in mathematics from University of Tunis, a master's in Operational Research (2000) from National Polytechnic Institute of Grenoble, and a PhD in Computer Science and Automatic (2004) from the University of Montpellier 2. As a scholarship holder of the French industry minister, he spent four years working on research and development projects at Alès School of Mines. assistant professor at the Department of Computer Science of the University of Lyon 2 from 2003 to 2004, he is currently assistant professor at the Pharmacy faculty of the University of Lille 2. His main fields of interest are: Modeling and verification of Discrete Processes, Conceptual Structures for Knowledge sharing, Graph-Theoretic Algorithms, Clinical decision support systems, human-computer interaction, Medical Informatics with special emphasis on Formal Methods.

François Prunet is professor from the University of Montpellier II. He became engineer from the ENSEEIHT in 1967 and obtains a PhD in 1972. His research concern discrete events systems and hybrid system applied in production systems engineering and automation domains. He was head of Procution systems Team from the Laboratoire d'Informatique et de Microélectronique de Montpellier (LIRMM/CNRS). He was one of the creators of the Grafcet (FCCS). He published several articles about control command, Hybrid system modeling and simulation (Batches Petri Nets) and focus currently on decision-making process support for production systems engineers. He was responsible of more than 25 PhD students in these domains.