Shareable descriptions of geographical data production processes

Bénédicte Bucher[1], Sandrine Balley[1], Didier Richard[2], Gilles Cébelieu[2], Jean-François Hangouët[3]
[1]COGIT Laboratory, [2]Projet Diffusion des Données, [3]IGN Quality Dpt Institut Géographique National, 2 avenue Pasteur, F-94 165 St Mandé Cedex
Tel : 33 1 43 98 80 03 ; Fax : 33 1 43 98 81 71
{benedicte.bucher, sandrine.balley@ign.fr, didier.richard, gilles.cebelieu, jean-francois.hangouet}@ign.fr

SUMMARY
Process description, once handled by complex knowledge engineering techniques, is becoming more accessible thanks to Model Driven Architecture techniques such as UML2.0. Meanwhile, the need increases to describe the production and transformation of geographical data through implemented tools. Firstly, documenting lineage metadata requires references to shared descriptions of production processes. Secondly, processes are themselves resources to be managed, e.g. catalogued. This paper reports on work in progress to describe low level data production processes within the French National Mapping Agency.

KEYWORDS: *Lineage information, Process description, Workflow management, Activity*

INTRODUCTION
Dating back to early Artificial Intelligence reasoning systems to more recent applications in the business to business (B2B) and enterprise application integration (EAI) areas, describing processes has gained more and more interest from the scientific community. With the improved representation of processes in UML2.0, this field, once reserved to knowledge representation specialists, should even become more widely accessible (OMG, 2003).

This paper describes a work in progress to improve the description of production processes within IGN, the French national mapping agency. It is mainly performed at the COGIT laboratory and benefits from collaboration with people from other departments of IGN, notably data diffusion, sales support and quality management. The first application of this work is the documentation of lineage information in data metadata. The resulting lineage metadata will have the form of detailed descriptions of geographical data production processes and will complement quality measurements that cannot cover all aspects of data quality. The first section details the context and objective of this work. The second section depicts related works. The next sections describe our model and a prototype application.

CONTEXT AND OBJECTIVE

Context
Descriptions of IGN data production processes exist, such as project management plans, analysis documents supporting software development, production specifications written at the conception stage, user guides, specific forms designed to control a process running, or log files. Most of these descriptions are textual documents that are little distributed and difficult to read. Log files are seldom archived and never shared with users of the produced data. A model to structure the description of production process is proposed in the ISO19115 model for lineage information but is somewhat loose. Lineage information is composed of *LI_ProcessStep* objects and *LI_Source* objects. A

LI_ProcessStep has free text properties, *description* and *rationale*, plus a *dateTime* and a *processor*. A more formal model is needed to build unambiguous and rich lineage information, as detailed hereafter.

A common vocabulary should be used to describe process steps. This could be a process types register. Theoretically, such a process types register should reference not only generic processes like "restitution" but also how these processes are carried out in various contexts, for instance depending on available source data or software.

Besides, describing a process type is not enough to account for its behaviour and qualify the resulting data. During a particular realisation, specific behaviours occur that originate specificity within the data themselves. These specific behaviours may be far-reaching consequences of the process strategy. For instance some generalisation processes may be said to be non predictable because they rely on a contextual strategy. These specific behaviours may also be linked to the "processors", implemented tools or human operators, and their various ways of doing similar things. Hence, in lineage metadata, a process step should refer to a process type description, like "*foreseen effects*: dead ends shorter than 50 meters are removed", and to a process realisation description, like "*observed effects* : 7 dead ends removed".

Lastly, the acquisition of production process description will probably be as distributed as the production steps are. An application to acquire complete, homogeneous and consistent lineage information is thus necessary.

Objectives

Our objective is to propose a model to describe geographical data production process types and realisations. Such descriptions may be used to infer content and quality information about of the resulting data. An application based on this model will support the distributed documenting of lineage metadata, their storage and their browsing. This model should be flexible enough so that in the future we may extend it to build other process management applications. The following section analyses related works that can be used to define our model.

RELATED WORK

Workflow management

Formalisms are proposed to define, implement and exchange business processes, for instance in the B2B or EAI areas. They may concentrate on the description of actors interaction at an abstract level, e.g the Business Process Modelling Language (BPML) and UML, or on the implementation based on Web services, e.g. BPEL4WS (BPMI, 2001)(OMG, 2003b). So far, there exists no consensual language nor conceptual model for describing processes. The OMG group has released a request for proposal about a Business Process Definition meta-model to unify existing languages (OMG, 2003). Figure 1 is an attempt to organise concepts common to most business process description languages. Operations are performed by performers, human or machine, to change the state of objects or to produce signals that will trigger other performers to perform other operations. These may be calculation, communication or scheduling operations. Performers may also be manipulated.

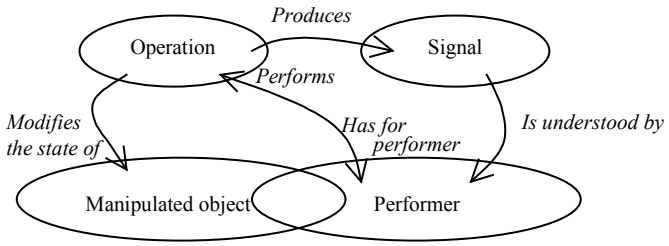

Figure 1. Attempt to organise concepts common to most business description languages.

Models alternatively focus on the behaviours, i.e. the operations on Figure 1, or on the affected objects, i.e. the manipulated objects on Figure 1.

The first type of model relies on the concept of activity, which is a unit of work. A complex activity is a set of sub activities organised after a control flow. The control flow schedules the data flow and message flow from outputs to inputs. BPML defines complex activity types, e.g. "Choice", after a generic control structure. An activity may be triggered by a message sent to one of its inputs or by a modification of its running context represented as a set of objects. The notion of "grounding" refers to implemented tools that support the activity, i.e. implemented performers. Business process groundings often are services. This technology is already being applied in the field of geographical information. (Brox., 2002) propose a very global model for a geospatial data infrastructure in Northrhine-Westphalia which includes business process descriptions.

The second type of model relies on the concept of "transition" between two states of manipulated objects. A "collaboration" groups transitions between states of the domain. In the UML state diagram, a collaboration is described as a scenario, a performer as an object, a manipulated object as an automate (OMG, 2003). Collaboration between agents is used for instance to organise generalisation operations performed by topographical objects on themselves (Ruas, 1999).

We can learn several points from these models. A process should be described independently from the tools that support it, i.e. its description should include a platform independent part. Behaviour-centred descriptions are not shareable as such. To be non ambiguous, they must be associated with the description of manipulated objects. Typically, to discover and use a Web service, the user must somehow understand the meaning of the data exchanged in the messages between the user and the Web service. This meaning may be conveyed in the message name, e.g. "sendMyName" or "getPrice", or in the type name, like "name". For Web services to be truly shared, they must be associated with ontological definitions of the objects they manipulate, like a tourism ontology. Lastly, what makes process description especially difficult seems to be the description of synchronisation: how to trigger behaviours or transitions.

Task planning

In symbolic Artificial Intelligence, reasoning processes have first been represented as mere sets of rules until it proved more efficient to organise these rules on a certain logic that reflects the decomposition of an initial task in subtasks (Clancey, 1985). Efficiency referred here to the ease of maintenance of expert systems.

The highlight was then put on building modular and shareable descriptions. To achieve this, (Gomez, 1999), among others, specify the lesson learned in workflow management : not only must behaviours and manipulated objects be described but their descriptions should not be melted. The notion of "role" introduced by (Marcus, 1988) is often used to describe behaviour variables as domain-independently as possible. An example of a behaviour variable is a classification criterion.

These principles have been used in the ESPRIT project KADS to propose a methodology to design knowledge-based systems (Schreiber, 2000). (Bucher, 2003) applies these principles to describe geographic application patterns through a tasks and roles based model, the TAGE model.

Besides, the objective of symbolic IA has shifted from building a machine that reasons, to building a machine that helps a human to reason, and to building machines that reason together. Hence, important tasks are communication and negotiation to ensure collaboration.

We can list the following interesting clues from this research. Again, synchronisation knowledge appears to be a crucial element. Again, to be shareable, behaviours description, i.e. problem-solving models, must be associated to domain ontologies. Moreover, the description of behaviour variables as domain elements must be as generic as possible for a process description to be as much reusable as possible. This description often relies on roles. From our work on TAGE, we have learned another clue concerning the reusability of process type descriptions. If a task is reused as a subtask in the decomposition of another task then it should appear in the decomposition as follows :

– with a reference to its generic description,
– with the specifications of its inputs and outputs in the current decomposition.

Last, what makes the description of reusable processes difficult is to describe and use the relationships between descriptions of a process and another more specific process.

PROPOSED MODEL

This section first underlines the specifications of our model and then reviews how a process is described in our model.

Requirements

Description of manipulated objects is handled in data metadata and in specific vocabularies used in production units. The tool description model is the subject of Yann Abd El Kader's PhD work at the COGIT laboratory. He adapts the OWL-S model, dedicated to Web services, to describe functions available in IGN software resources, which are not Web Services. The relationship between both works is that the realisation of some processes consists in applying an implemented function.

Our objectives are less ambitious than those of the literature models. We concentrate on describing processes that run programs to yield geographical data as an output rather than on processes that are made up of interactions between people. Neither do we describe IGN organisation, nor integrate people's beliefs, wishes or culture, nor describe multi-agent collaboration. As far as synchronisation is concerned, we will first limit the model to simple synchronisation as proposed in most procedural languages (*if*...*then*, *repeat*…). We do not wish to automatically implement any kind of process after its abstract description, so we need less formalisation than meta-models like UML2 do.

Eventually, the main objectives of our model are the following.

It should support the creation of generic process types descriptions that may be reused to describe more specific processes or realisations of processes. More precisely, each process type description should be reusable in a more specific description unless it is specific enough to correspond to only one possible realisation. And descriptions may be created dynamically.

We wish to adopt an iterative approach, i.e. to propose a prototype and then improve on it with the help of experts and experience. We expect these experts to be willing to interact not only with a user interface but also with the model itself. Hence the first versions of the description model should be as readable as possible, so that we can work with experts directly on the model and not systematically through a user interface. The first versions are dedicated to documenting lineage information. Yet, this model should be extensible so that we can further enrich it with elements required by process management applications.

An object representation of activity diagrams

Since activity diagrams are more familiar to most experts we work with than tasks models or state diagrams, we base our model on UML2 activity diagram stereotypes where a process type is described as an activity. Thanks to not-too-ambitious objectives as exposed in the preceding section, we can drastically simplify UML2 to work on a much more readable model. We keep some core elements, sometimes modify them. For instance, in UML, the class *Action* addresses executable units of work, possibly the invocation of an *Activity*. In our model, we interpret the concept of process as a "a set of potential process executions in a certain context". Thus, an executable process is of the same type as a non executable process, only its set of possible executions is more limited. For instance, a process that appears in the decomposition of another process is an *Action* in UML2 whereas, in our model, it is of the same nature as a stand alone process, only it refers to executions that take place in a more restricted context.

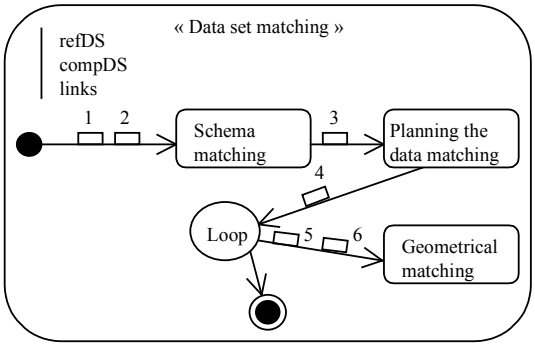

Figure 2. Graphical representation of the activity "data set matching".

Figure 2 illustrates a summary activity diagram of the *Activity* "Data set matching". Rounded rectangles are activities; arrows are edges between activities, possibly supporting object flows (numbered rectangles). The circles are control: activity starting point and end point, loop or other control structure. The terms "refDS" – for “reference data set”-, "compDS" –for “compared data set”- and "links" are the names of the process variables.

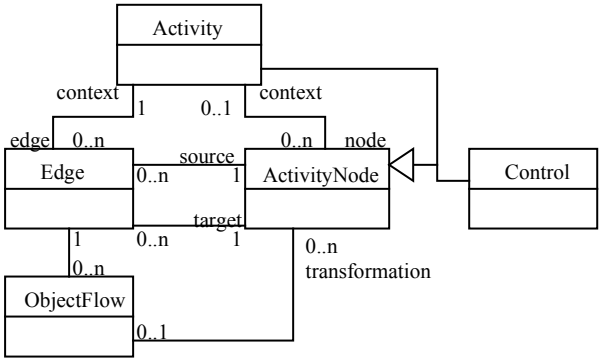

Figure 3. Class diagram of the main objects we use to describe an activity.

As illustrated on *Figure 3*, the object-oriented description of a process naturally relies on *Node* and *Edge* classes. Nodes may be *Activity* or *Control* objects. Edges may carry control messages and objects. An edge that carries an object may also carry transformations on it that are activities with a single input and a single output. The carried object may come from the source activity of the edge or

it may be selected from the context of the edge. Edges are described in the order in which they should be navigated over. *Figure 4* represents some elements (not all) of the object representation of the activity data set matching as performed in the GeOxygene platform..

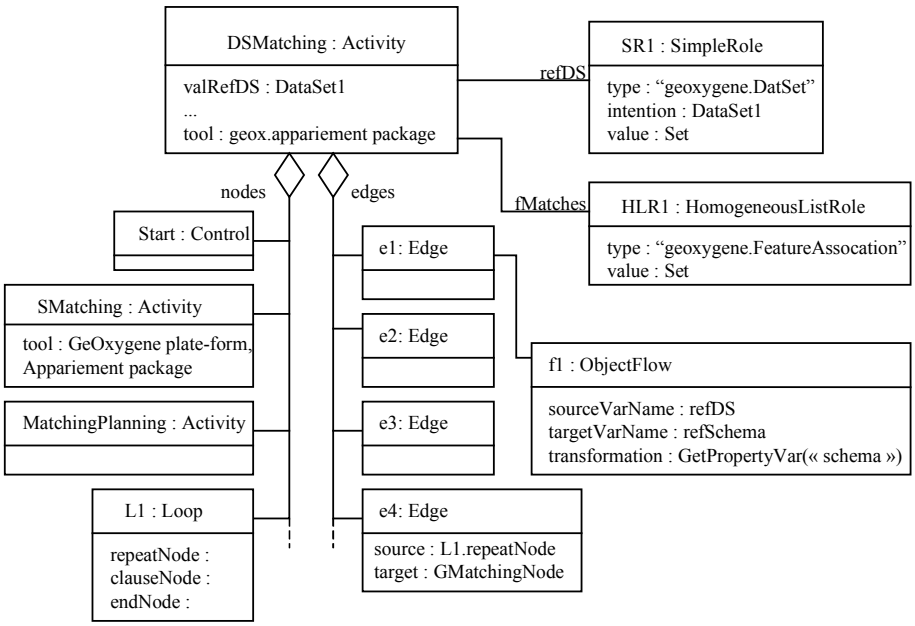

Figure 4. Instances diagram of the object representation of the process Data set matching.

The class *Role* describes the Activity variables. A *Role* instance describes a set of possible values. Several subclasses of *Role* are defined to describe all possible types of variable, like *SimpleRole*, *ListRole, HomogeneousListRole* or *TreeRole*. We also adopted the following convention: in case a variable called *myVariable* has for value a *SimpleRole*, e.g. the variable *refDS* on *Figure 4*, then a variable called *valMyVariable* may be added to the Activity description to refer directly to the intentional description of this *SimpleRole* value set, e.g. *valRefDS* refers directly to *DataSet1* on *Figure 4*.

Process description creation and reuse

Before describing how our model supports the reusability of descriptions, we summarise the successive temptative models that have led to it. We have tried out several models to meet our requirements, especially the description of the relationship "is more specific" between process descriptions, relationship on which reusability is based. In our model, a process is more specific than another if its set of potential executions is included in the other's.

Describing every process as classes extending an abstract class *Activity* seemed adapted to applications dedicated to manage a same process. It was not adapted to applications managing different types of processes, like the dynamic creation of process descriptions. Moreover, the relationship "is more specific" between process types cannot be supported by class extension. Indeed, a process may have multiple relationships of the form "has a set of possible executions included in the set of possible executions of".

Describing every process as instances of a generic class *Activity*, e.g. tasks in TAGE, either lacked expressiveness or yielded verbose descriptions.

Recently, we tried out a dual representation where a process was described through a class extending an abstract class *Activity* and through an instance of a metaclass *ActivityType*. This revealed a heavy approach where two different models were to be managed in a consistent way.

We have chosen a more flexible and simple way, which is introduced in the preceding section. We describe hereafter how it supports the reusability of descriptions. A process type is described by an instance of the class *Activity,* or of a class extending *Activity*. This allows specific procedural knowledge to be integrated in the description of a process type, in the form of class methods. For instance, we define the class *GetPropertyVar* to describe the process of getting the value of a property. An instance variable of *GetPropertyVar* is the name of the property. In the example on *Figure 4*, the object carried by the ObjectFlow f1 is the value of "refDS" in the activity DSMatching. It is transformed through an instance of *GetPropertyVar* with property name "schema". Hence, the schema of the reference data set is transferred to the target variable.

Much care was put on describing the "is more specific" relationship between two process types, i.e. between two activities. It may rely on extension between *Activity* class definitions. It may also rely on two properties of an *ActivityI.* The *roleSpecification* property expresses how the roles described in the class definition are specified in the current instance. The model for this property is detailed in the next section. The *specification* property refers to a more generic Activity and to *roleSpecification* applied to the roles of this more generic Activity. An activity may have several *specification* properties. This is illustrated on *Figure 5*.

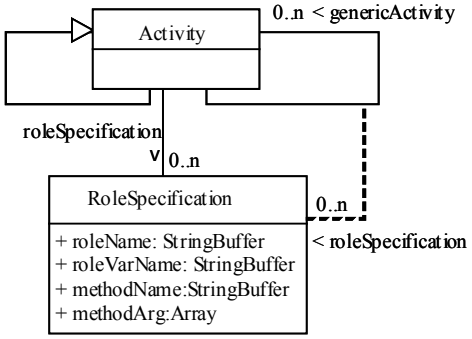

Figure 5. The description of the "is more specific" relationship between activities.

Roles specification

A role specification is described by the name of the role, the name of the role inner variables in case it is a structured role, e.g. the variable "*Leafs*" in *TreeRole*, the name of a specification method, the arguments of this method. The methodName may be either of the form "myPackage.myRoleStructure.mySpecificationMethod" or "myDomain.myType MySpecificationMethod". For instance, a user may reuse the DatSetMatching description and specify, in his description, that he works on datasets with linear elements. This may be done by writing a specific method in the *DataSet* class like *specifyGeometriesType*. Or, if the class *LinearObjectDataSet* exists, it may be done by using a generic method in the *RoleSimple* class like *specifyType*.

Describing the resulting data

Experts usually account for a result either by stating that "this process usually has such-and-such effects" or by stating that "this process works like this", where the described behaviour has immediate effects on the results. Two properties have been added to an Activity to enhance the description of the resulting data: *effects* and *mechanisms*. *Activity effects* are composed of a description and a confidence, e.g. (every building gets squared; sure), (missing z value are often set to 0 ; probably). *Activity mechanisms* are composed of a performer, a description and a confidence, e.g. (the tool Geoconcept; rebuilds the topology; hypothesis).

Realising a process

In our model, every *Activity* has a method *realise*, which returns a *ActionPlan*. This *ActionPlan* is a list of actions a user should perform to realise the activity, like "Put these data in this file", "Click on the command File/New/Project on the user interface", "Write the command … ". Three types of *realise* methods must be distinguished.

When the *Activity* can be realised through classical java procedures, the return *ActionPlan* is null and the method actually performs the activity. This is the case when the *Activity* inputs and outputs are java objects and when the behaviours are encoded in existing API or in a tool that has a java compliant programmatic interface.

When the *Activity* corresponds to the use of an implemented tool that does not have a java compliant programmatic interface, the *ActionPlan* is a list of possible preprocessing steps and of actions on the tool user interface.

When the *Activity* is a complex Activity, the method *realise* navigates its graph of Nodes. At each Node, it invokes the *realise* method. If the return value is non null, it waits for the user to perform the instructions listed in the current *ActionPlan* to navigate the rest of the graph.

In most cases, writing these *realise* methods rely on the tool metadata model. They have not been written in the current prototype because documenting lineage metadata does not require to realise described processes.

CONCLUSION

This paper describes how we build a generic model to describe low level data production processes within IGN. We propose a first version of our model and then refine it with the help of people who use it. As compared to models found in the literature, our model is much simpler and readable. This is possible because it meets so far less ambitious objectives than existing models. We focus on building shareable and reusable process description to support the documentation of lineage metadata. Yet, we remain close to UML2 so that in the future we may refine our model to meet other process management objectives.

A first prototype has been implemented. Its current function is to help users specify unambiguous values for their activity roles by referring to an existing glossary or model of their domain. The interface is so far in French because it is intended for internal use. So far, we have handled the case where the domain model is a set of java classes, as is the case at the COGIT laboratory. The user can load any class description, create a new object and browse its attributes and relations to specify them through graphical components only. *Figure 6* presents the edition of an Activity object and the possible ways to define a value for a property -here an *ActivityNode*- : the user may create either create a new object, relying on the constructors or other methods, or create a reference to an existing object.

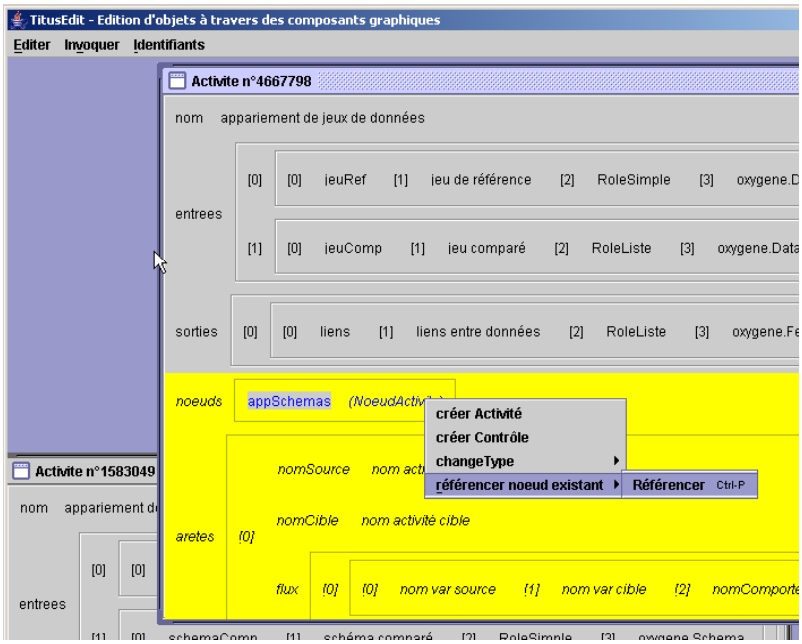

Figure 6. Snapshot of our prototype application illustrating the edition of objects through graphical components.

The system records the specifications performed by the user through the interface. So far, the user may browse attributes, invoke any type of method, create and reference objects. Expression of constraints on literal values is not currently managed.

On-going work concentrates on editing process description through an activity diagram like display of the model relying on the GEF framework. The creation of these descriptions will be tested with users from the COGIT laboratory and from some IGN production units.

BIBLIOGRAPHY

Brox, C., Bishr, Y., Senkler, K., Zens, K., Kuhn, W., 2002, Toward a geospatial data infrastructure for Northrhine-Westphalia, Computer, Environment and Urban Systems 26, 2002, pp.19-37.

Bucher, B., 2003, Translating user needs for geographic information into metadata queries, 6th AGILE Conference, France

Business Process Management Initiative, 2001, BPML1.0, Business Process Modelling Language

Clancey, W., 1985, Heuristic Classification, University of Stanford report STAN-CS-85-1066

Gómez A., Benjamins R., 1999, Overview of Knwoledge Sharing and Reuse Components : Ontologies and Problem-Solving Methods, Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods, Stockholm, pp. 1.1-1.15

Marcus S., 1988, Automating Knowledge Acquisition for Expert Systems, Kluwer Academic, Boston

Object Management Group, 2003, Business Process Definition Metamodel, Request for Proposal

Object Management Group, 2003, UML 2 Superstructure Final Adopted Specification http://www.omg.org/cgi-bin/doc?ptc/2003-08-02

Ruas, A., 1999, Modèle de généralisation de données géographiques à base de contraintes et d'autonomie, Thèse de doctorat en Sciences de l'Information Géographique de l'Université de Marne la Vallée

Schreiber, A. Th., Akkermans J. M., Anjewierden, A.A., de Hoog, R., Shadbolt, N. R., Van de Velde, W., Wielinga, B. J., 2000, Knowledge Engineering and Management, The CommonkADS Methodology, MIT Press