Building Information System Requirements Using Generic Structures

G. Grosz*

INRIA, Domaine de Voluceau - Rocquencourt BP 105, 78153 LE CHESNAY Cedex FRANCE

Abstract:

We present generic knowledge to speed up the construction of information system requirements and more importantly the behavioral part of entities. Our solution is based on the hypothesis that generic structures (independent of a particular application) can be associated to classes of real world phenomena. Building information system requirements using such structures means to instantiate those structures to the current context. The designer no longer has to redo the conceptualization effort, he can concentrate on the perception of the reality being described. In order to use those structures and to represent the generic knowledge, we present a design model process based on the triplet formalism <situation, decision, action>.

1. Knowledge used during the requirements engineering activity

The Information System (IS) design process is classically divided into two steps as shown in figure 1. The first step leads to the construction of the so-called conceptual schema, starting with the analysis of the real-world phenomena and the users needs. This conceptual design step corresponds to the "Requirements Engineering" (RE) activity, RE refers to the activity by which a designer acquires knowledge on the application domain and abstracts the conceptual definition of the future IS. RE is guided by the information needed from the users.

The second step of the design process uses the conceptual schema as the starting point. It leads to the construction of the internal schema. Unlike the conceptual schema, this schema takes into account technical constraints, material features, etc..

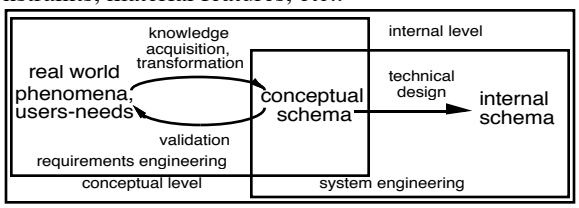

Figure 1: The global IS design process

* Currently visiting the University of Texas at Arlington, Department of Computer Science Engineering, Box 19015 ARLINGTON, Texas 76019-0015, USA email: grosz@cse.uta.edu

These two steps correspond to the abstraction levels defined in the ANSI/X3/SPARC report [1], namely the conceptual and the internal level.

The design process is a creative process primarily supported by the designer. Design methodologies - whether functional (i.e. SADT [2]) or systemic (i.e. REMORA [3]) - are limited to define a sequential list of tasks. The different tasks are not formally defined, they are organized in a linear manner, this does not reflect the practical reality of design.

Therefore, it is necessary to improve our understanding of the acquisition/transformation task. Additionally, we must describe the various kinds of knowledge involved during the design process and define how this knowledge can be represented.

We believe that such representation is mandatory for the emergence of the next generation of CASE tools. These tools will be both active and intelligent, capable of automatically guiding the design process and possibly automating some of the creative tasks which are today totally supported by designers.

Current tools are based on research conducted around the concept of **product** [4]. These tools focus on results obtained at the different stages of the design process. They are product managers: they help to capture the specifications with user-friendly graphical interfaces, store the specifications in a database, validate, and document them. This kind of tool only partially helps in the construction of the specifications. This is because none of them has been integrated the approach, the process which leads to build specifications. This integration will be possible only if both the different design tasks are precisely described and the scheduling of those tasks is represented.

We believe that design tasks can be based on generic structures independent of any particular application, and that these structures can be re-used in the development of different projects. Our hypothesis is that there exists classes of similar real-world phenomena which are described using identical structures. A generic structure describes either the static and behavioral properties of a class of phenomena. Designing an application can be seen as the recognition of these phenomena and the instantiation of the associated generic structures.

The paper is organized as follows : section 2 describes how we organize the first step of the requirements engineering activity. We divide this into two phases. The first

one leads to the elaboration of an intermediary descriptive schema. During the second phase, one transforms the descriptive schema into a conceptual schema. For each phase, a specific kind of generic knowledge is involved. In this paper, knowledge involved in the first phase is described in detail. In section 3, the formalism used to express the generic knowledge is presented, namely the triplet <situation, decision, action>. The different concepts used to build a descriptive schema are introduced in section 4. A presentation of generic knowledge with examples is given in section 5. Section 6 illustrates the use of such knowledge through an example.

2. A specific organization of the requirements engineering activity

The RE activity can be divided into two phases : acquisition and transformation (figure 2). Acquisition corresponds to the understanding of the universe of discourse, the selection of relevant phenomena and the description of this phenomena into an intermediary descriptive schema. This schema is used as a draft for building the conceptual schema. During the transformation phase, the descriptive schema is completed and transformed, using the conceptual model, into the conceptual schema .

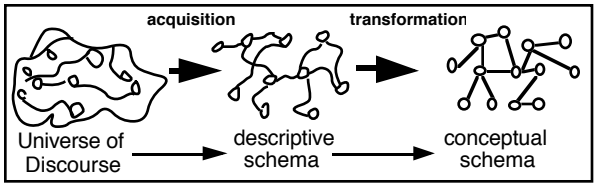

Fig. 2: Our view of the requirements engineering activity

For each of these phases, one must formalize and represent specific knowledge:

- Domain knowledge for the first phase
- Model knowledge for the second phase

Domain knowledge indicates the method for describing real-world phenomena using a model. The study of specifications describing different applications (e.g., library, hotel reservation, stock management systems), leads us to emphasize the fact that many different phenomena can be described in the same way using the same concepts. The labels are to be different, but the behavioral structures are to be the same. For example, the behavioral structure for checking out a book in a library is very similar to the structure which describes the behavior for plane reservations, train reservations and car rental.

Domain knowledge corresponds to the use of generic structures describing very general classes of real-world phenomena; it is described in greater detail in this paper. We concentrate on the identification of the classes of phenomena and on the definition of the corresponding generic structures which allow them to be represented in a way that facilitates their re-use and adaptation to a specific context.

Model knowledge describes pre-defined transformation schemas from a model to another. We define classes of general structures as descriptive schema sand establish their correspondence in terms of the conceptual model. For instance, one can pre-define the set of all possible transformations for a binary association between two entities in the descriptive schema into different objects schemas structures using a conceptual model such as O* [5]. Depending on the cardinalities of the association, the transformation schema is different but can be described a-priori. This kind of generic knowledge is detailed in [6] and uses generic structures by taking into account statical as well as behavioral properties.

3. The process model

The knowledge involved during the acquisition/ transformation activity is tightly linked to the process by which an input is transformed into an output (e.g., (1) the recognition of a real-world phenomena is transformed into a sub-descriptive schema; (2) an entity of the descriptive schema is transformed into different objects schemas). In order to represent this knowledge, we defined a process model based on a simple formalism. We propose to describe all activities of the requirement engineering process, whatever its nature (abstraction, validation, transformation...) and its granularity, by using the triplet formalism:

<situation, decision, action>

A situation corresponds to a piece of current specification. It can be elements of the descriptive or the conceptual schema.

A decision is the intention of the action. Depending on a particular situation, it determines the appropriate action to perform on the current specification.

An action describes the transformation induced from the decision taken in a given situation.

Figure 3 illustrates the relationship between the elements of a triplet.

- a situation *implies* a set of decisions;
- taking a decision *triggers* one or many actions;
- an action *modifies* the current specification and thus, the situation.

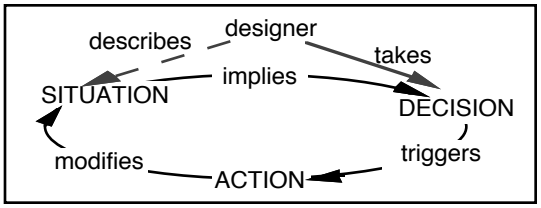

Figure 3: Relationship between the triplet's elements

The designer initiates the loop by describing an initial situation. The loop ends when the current specification does not match any situations described in the different triplets.

All knowledge required for the RE activity is represented using the triplet formalism.

A triplet can be seen as a rule (a premise implies a conclusion) where the premise has two distinct parts: situation and decision. This dichotomy is important

because it allows us to describe the design process at a microscopic level of detail. It emphasizes the decisional aspect of the design process. Using the triplet formalism, one can describe precisely and in great detail the different possible transformations in a given situation depending on decisions.

4. The descriptive model

The generic knowledge presented in this paper is limited to "domain knowledge". Generic structures are expressed using the different concepts of the descriptive model. We briefly describe each of these concepts. A detailed description can be found in [6]. The descriptive model is based on three concepts : event, actor and entity.

Every instance of the descriptive model concepts (the entity "car", the actor "subscriber") has a name ("car", "subscriber"), a wording expressed in a free text and a set of properties.

4.1. Event

An event defines a state change in the IS. This concept includes both the cause of the change and the action induced by the change. An event corresponds to a process and its trigger. Every state change is an event. For instance, every state change of a book in a library is described by different events: "check out", "return", etc..

In an event's description, it is important to precisely define the triggering actor or entity. The event wording contains the description of the occurrence condition and all of the induced actions with their conditions.

4.2. Actor

The actor's concept exists in the real-world. An actor initiates and controls the execution of events. An actor is supposed to dispose the required resources (energy, tools, space, or informations) and conduct the mission he is responsible for. An actor has the ability to issue messages which must be stored by the IS. This concept allows one to describe the people or the machines which interact with the IS. An actor may trigger one or more events. He contributes to the modification of entities and other actors described in the IS.

4.3. Entity

An entity represents an abstraction of a real-world phenomenon, which may or may not have a physical existence and has a finite lifetime. An entity is used to describe a class of phenomena for which the IS must provide information.

4.4. Trigger and target

Events are triggered either by actors or by entities. The concept of trigger is introduced in order to generalize these two concepts. It allows one to generally describe "something" which triggers an event. We also introduce the concept of target which enables us to generalize the concepts of entity and actor since both can be modified by event. These two concepts facilitate the description of generic knowledge presented in section 5.

4.5. Graphical representation and sample

Every concept of the descriptive model has a graphical description shown in figure 4.

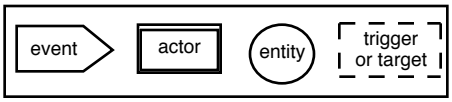

Figure 4: Graphical representation of the concepts

Figure 5 presents a sample of what a descriptive schema can be. It expresses that the actor "customer" triggers the event "order" on the entity "product'. The actor "representative" is related to the entity "company car" and also to the actor "customer". The event "supplying order" is an event triggered by the entity "product" and modifies the actor "supplier". A "product" is related to a "warehouse". Dynamic relationships involve an event while static ones do not.

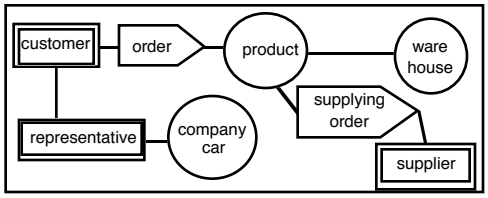

Figure 5: A sample of a descriptive schema

5. Generic domain knowledge

In this section, we present some examples of generic domain knowledge expressed using the triplet formalism described in section 3. This knowledge allows one to build descriptive schema by using pre-defined generic structures.

The generic structures are the expressions of statical and behavioral properties which always exist for classes of real-world phenomena. These classes are characterized with simple words and can be easily detected in the real-world. The generic structures are the action part of triplets. A triplet expresses a contextual link between a characteristic of some application domain element and its generic representation in a descriptive schema.

The first triplet presented in this paper describes the use of resources. A resource is an entity which is shared, required and available in a limited quantity for the management it is involved in. In a library, for instance, the entity "copy book" is a resource for the "loan" process. In a personal computer network with a shared printer, the entity "printer" is a resource while the process is the "printing of a document".

When a resource is available, it can be used in a consuming process when requested by a consumer. As soon as it is consumed, it becomes unavailable for the other possible consumers because a resource is at the exclusive use of the consumer who obtained it. In a commercial company, a product is a resource for the "buy" process, when a consumer buys a given product, it becomes unavailable for the other consumers.

When a resource becomes unavailable (curative

management) or will become unavailable (preventive management), it may be useful to ask more from the supplier. For instance, when the inventory of a product becomes low or zero, an order may be automatically sent to the supplier. Fulfilling the order may have no meaning if the availability of a resource is set at the creation time (as for a printer in a local network).

A resource may be created (its entry in the system corresponds to a creation). Its supplier can be internal to a system (a company building products) or external (as in a super market). Complementary, a resource may be deleted.

The triplet associated to the general management of a resource is presented in figure 6. The situation is an entity; the decision is to characterize this entity as a resource; the action part makes use of generic structures. It describes, on one hand, the behavior of a resource as we described it in the previous paragraphs and the minimal set of properties that the entity must have.

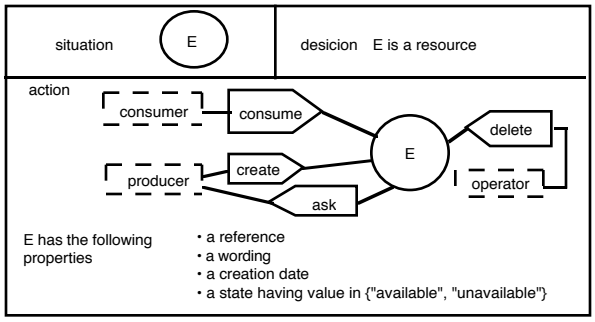

Figure 6: the general triplet for a resource

The study of the different types of resource management leads us to define different categories of resources. Some are consumable (for instance, the product bought by customers in a stock management system); among those, some are perishable (fresh products in a grocery) and other are non-perishable (the hardware or electronics). There are also re-usable resources (for instance the cars in a rental company). In this case, one can distinguish between renewable (books in a library or cash in a bank) and repairable resources (cars from a rental company or planes from an airline). Of course, a resource can be characterized by more than one of the mentioned characteristics. Figure 7 shows how the different characteristics are organized in an "is_a" hierarchy. The ⊗ symbol, borrowed from the NIAM notation [7], expresses that the two classes are disjunctive.

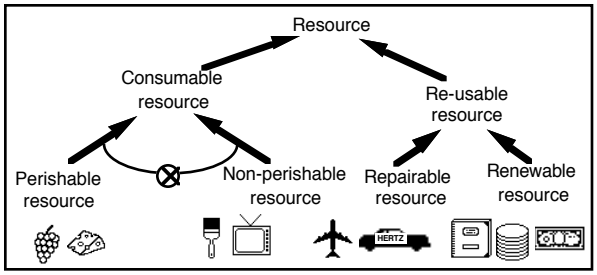

Figure 7: hierarchy among resources

For all classes, we defined a triplet which describes the specific properties of the behavior using ad hoc generic structures. All of these triplets are detailed in [6]. The way we build them is the same as the one we used to build the general one (figure 6), (i.e., to define the particular behavior and properties and express it using generic structures). Each class of the hierarchy inherits the behavior and properties of its super classes.

The second triplet defines how to manage the resource requests in a queuing list. The purpose is to define how the system will react (1) when it cannot fulfill a resource request and (2) what it should do when a resource becomes available and there exists a request concerning it. In a commercial company, back-order management is often used. As soon as a product is available, the system checks if a back-order exists concerning this product, and if so, it tries to satisfy as many requests as possible.

We assume that the system has the possibility to check the availability of resources. A request is put in the waiting list when all (or part of) the required resources are not available. Thus, the system must:

- store all the information which will be necessary for the recovery of the the request when it will be possible; that is "who made the request ?" (the consumer) and "what is the request about□?" (the requested resource(s)).
- know the details of the strategy which will be applied for the recovery of the waiting request; the recovery strategy is dependant on the application, the designer will have to define it.

When an instance of the entity "resource" is subject to an event adding availability, if there is some waiting request for this resource, the event "recovery of waiting request" is triggered. This event tries to satisfy as many back-orders as possible. It corresponds to:

- a modification of the concerned request, their states change from "waiting" to "satisfied"
- a modification of the allocated resources' state, it changes from "available" to "unavailable"
- a message sent to the associated consumer.

The triplet using the generic structure is shown figure□8. The situation is an entity, the decision is to characterize this entity as a resource and to manage the waiting request.

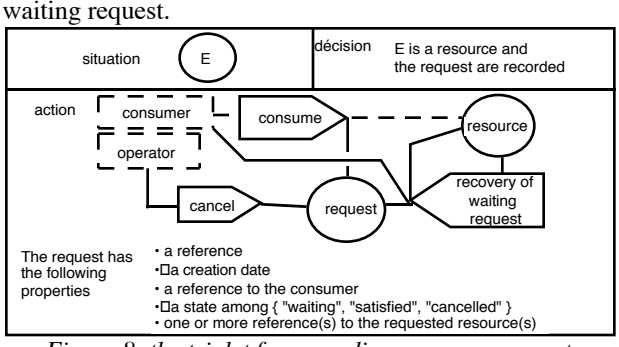

Figure 8: the triplet for recording resource requests

6. Using generic domain knowledge

This knowledge must be used in a CASE tool library. A library completely changes the nature of the requirement engineering process itself: the designer does not have to express the real word phenomena using a conceptual model but only to identify those phenomena.

When the process is driven by a CASE tool, it is the tool's duty to use a matching algorithm between the existing specification and the situations described in the different triplets. When matching is found, the tool proposes to make the different possible decisions. When the designer makes the decision, the corresponding generic structure must be instantiated.

The instantiation algorithm has two steps:

1) set correspondence between the elements proposed in the generic structures and those already described in the current specification (names can be different but functions are the same);
2) refine the descriptive schema by adding (if necessary) the generic structure elements which did not meet correspondence; these may have to be renamed taking into account the current context.

The process underlying the use of the triplet is a process of re-use. This process is well adapted to the requirements engineering activity. This activity is not guided by the goal (the conceptual schema) but rather the source (the requirements themselves). The re-use we propose does not force the designer to identify a goal but to identify the source then to re-use a previous process for the representation of this element. This kind of re-use is different from the re-use of software components or objects schema as experimented, for instance, in the ITHACA project [9]. In fact, these two kinds of re-use are complementary and should be combined in a single CASE tool.

In this paper, the use of the triplets is illustrated with the example of the loan management in a library.

Every sentence written in italics describes a dialogue between the tool and the designer.

In the following figures, all elements obtained in a previous step are greyed.

The essence of the activity in a library may be summarized as the following: "subscribers borrow books". This initial description corresponds to the descriptive schema figure 9.

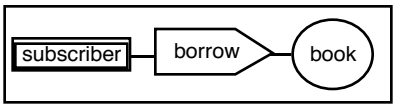

Figure 9: a first description of the library system

Using this first description, the tool asks the following question:

- *tool: is entity "book" a resource ?*
  designer: YES

The triplet described in figure 6 is used with the entity book in place of the situation. During the instantiation of the generic structure, the correspondences expressed by the designer are the following:

- event "consume" with the event "borrow",
- the trigger "consumer" with the actor "client".

During the refinement step, the "supplier" and the "operator" described in the generic structure are identified as the "librarian". The event "create" corresponds to the addition of book in the library, its name is changed to "insert". The events "delete" and "ask" correspond respectively to the the deletion of a book and to warn the librarian that a book is required. The descriptive schema obtained using the triplet is shown figure 10.

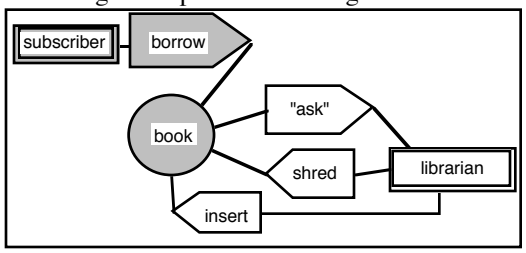

Figure 10: the descriptive schema after applying the general resource triplet

- *tool: do you want to manage the request concerning the resource in a queuing list ?*
  designer: YES

The correspondence phase leads one to recognize, in addition to the correspondence already mentioned (consumer/subscriber and consume/borrow), the "librarian" as the actor "operator" who cancels requests. The refinement phase leads to the addition of the events "cancel" and "recovery of waiting loan" and the entity "loan". The result is shown in figure 11.

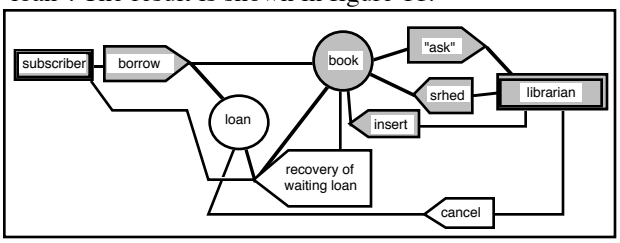

Figure 11: the descriptive schema after applying the triplet concerning resources request recording

Through this example, we have shown, starting from a very concise description of loans in a library, using two triplets, how we build the descriptive schema shown in figure 11. The process should be continued to transform the schema in a conceptual schema using what we call model knowledge.

7. Conclusion

We started the implementation of the domain knowledge using the shell of the OICSI tool [8]. OICSI is an expert system CASE tool with a user friendly graphical interface written in Objective C. The original rule base, written in Prolog, has been simply changed with the triplets proposed. Some minor modifications have been made in the graphical interface, taking into account the specific diagrammatic notations used in the

descriptive model.

In this paper, we presented a formalism which allows one to describe the knowledge used during the IS development process in a structured manner using triplets of the form: <situation, decision, action>. The triplet establishes a link, using decisions, between a situation within the current specification and an action on this specification. The triplet is more than a knowledge representation technique, it is the basic concept of a process model, this allows one to describe the whole IS development process as an evolutionary process describing the mutation of the specifications from fuzzy (the requirements) to formal (the conceptual schema).

In order to facilitate the communication between the end-users and the designers in the earliest step of the design development, we propose to use a simple and intuitive model called the "descriptive model". Based on three concepts, namely actor, event and entity, it allows one to describe in both graphical and textual ways the information flow within an organization.

On the other hand, we show that there exists reusable knowledge for IS design. Using the triplet, we are able to represent this knowledge. We distinguished two kinds of knowledge:

- knowledge related to the representation of real-word phenomena using conceptual structures (domain knowledge). This is the main issue of this paper;
- knowledge allowing one to transform a conceptual structure expressed in a given model into another model (model knowledge). This is not described in this paper, see [6].

Domain knowledge uses generic structures, each one dedicated to very general real-world phenomena. These structures are expressed in ways which facilitate their reuse and their instantiation to a particular context.

The generic structures we put forward do not cover every real-world phenomena. This study will be extended to other type of entities, actors and events. We are studying how generic structures could be discovered using a learning by example tool, with existing conceptual schemas as input.

Representation of generic knowledge is an innovative approach. It allows one to re-use the process by which specifications are built. It goes along on the same line as Reubenstien's works [10]. Coupling our work with the classical approach for re-using software components is a promising research direction. It allows one to cover the whole IS development cycle. The problem is to study how generic structures can match software components in order to build specific projects.

Acknowledgements : The author wishes to thank the INRIA for its financial support during his stay in the University of Texas at Arlington and Drs C. Kung and J. O. Smith for their fruitful comments on previous versions of this paper.

References

[1]: ANSI/X3/SPARC: "Interim report on data base systems", 1977.

[2]: D.T. Ross, K.E. Schoman: "Structured Analysis for Requirements Definition", IEEE Transactions on Software Engineering, Vol. 3, N°1, January 1977.

[3]: C. Rolland, O. Foucault, G. Benci, "Conception des systèmes d'information: la méthode REMORA", Eyrolles, Paris, France, 1987.

[4]: T.W. Olle, J. Hagelstein, I.G. Mac Donald, C. Rolland, H.G. Sol, F.J.M. Van Assche, A.A. Verrijn-Stuart: "Méthodologies pour les systèmes d'information", Dunod Informatique, Paris, France, 1990.

[5]: J. Brunet: "Modeling the World with Semantic Objects", Proc. of the IFIP Conf., Québec, Canada, October 1991.

[6]: G. Grosz: "Formalisation des connaissances réutilisables pour la conception des systèmes d'information", PhD dissertation University Paris 6, Paris, France, December 1991.

[7]: G.M.A. Verheijen, J Van Bekkum: "NIAM: An Information Analysis Method", in "Information Systems Design Methodologies: a Comparative Review", Proc. of the IFIP WG 8.1 Working Conf., North Holland (ed.), Noordwijkerout, The Nederland, 1982.

[8]: C. Cauvet: "Un modèle et un outil d'aide à la conception des systèmes d'information", PhD dissertation University Paris 6, Paris, France, November 1988.

[9]: B. Pernici, G. Vaccari, R. Villa: "INTRES : INTelligent REquirements Specification", Proc. IJCAI'89 Workshop on Automatic Software Design, Detroit, Michigan, USA, August 1989.

[10]: H.B. Reubenstein, R.C. Waters: "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", IEEE Transactions on Software Engineering, Vol. 17, N°3, March 1991.