SYNERGIE

Projet IGLOO

Litterature Survey On User Requirements Specification and Incremental Specification

Millestone: M.c.1.NSERC

by Stéphane Sotèg Somé,

Rachida Dssouli and

Jean Vaucher

Montréal

Contents

Overview

The IGLOO project M.c.NSERC [1] aims at using Scenarios for the expression of user's requirements, and to derive Object-Oriented design from these requirements. Scenarios are systems observable behavior that describe possible sequences of interactions between systems and their environments.

Interactive systems like communication systems have an observable behavior feature. Then users requirements for such systems are often given as a set of scenarios, each describing a possible sequence of interactions between the system and its environment. Each of these scenarios is an observable behavior of the system. Our objective is to develop an higher level language supporting such scenarios and permitting the derivation of complete and valid specifications.

Scenarios are often incomplete and contradictory. The language must then have an underlying formalism for verification and validation. It must also be possible to smoothly add new scenarios to an existing specification. The scenario language is therefore assumed being able to support incremen-

[1]This work was funded by La Chaire de Téléinformatique du Département d'Informatique et de Recherche Opérationnelle, Université de Montréal

tal specification. For an efficient conformity verification of the synthesized specification and the original requirements, some mechanism binding each scenario with the piece of specification which satisfies it must be included in the synthesis process.

From the above the key domains of the project M.c.NSERC can be enumerated as follow:

1. Users' requirements engineering,
2. Incremental specification,
3. Requirement tracability and
4. Real-time system design.

This survey concerns users' requirements engineering and incremental specification its goal is to review the actual situation of these two research domain.

The survey includes two parts. Part 1 review users' requirements engineering and part 2 incremental specification. A complete reference is provided at the end of the survey.

Part 1

Survey On User Requirements Specification methods

Software engineering was defined by Boehm (Boehm, 1976) as the application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them.

Several technics and methodologies for software engineering exist but all of these methods begin by a software requirements engineering phase which aims at developing a complete, consistent and unambiguous model of the software system. Subsequent phases to requirement engineering include the system design, system implementation and maintenance.

The role of software requirement engineering has been recognized as a crucial one because the majority of the software misfunction originate from this phase and these errors are the most difficult to detect and the most expansive to correct (Boehm, 1984).

Requirement engineering comprise the discovery, refinement, modelling and specification of the needs for the software system (Pressman, 1992). Its must be noted that the sequence of the above steps is not necessarily sequential.

The requirement discovery step consists on the elicitation of the software system characteristics expected by the user. The software characteristics include the function waited for the system and other non-functional requirements as performance, security or operational cost requirements. Requirement discovering results on a set of informal requirements directly obtained from the user.

The first set of requirements obtained often includes ambiguities and contradictions which leads to a misunderstanding of the needs. A step of refinement is often needed after the first analysis of the requirements set in order to obtain a complete and correct set of requirements.

The system modelling consists on the building of a general model of the software function, information flow and other characteristics. The software model shows *what* the system must do. A Model may assist the designer to have a better understanding of the system and serves as a basis for a further requirements gathering or for the software system prototype building.

A software requirements specification is produced as a result of the software requirement analysis phase. The requirements specification is the complete description of the software structure and function. Some methods produces verifiable specifications by using formal specification languages. Requirements specification further serves as a basis for the software system design.

Requirement specification is an abstraction and a formalization of the user needs and the intended software system. Several methods have emerged for requirements analysis such as Data Flow Diagrams methods, JSD (Jackson, 1983), SADT (Dickover et al., 1978), PSL (Teichroew et al., 1980), EDDA (Trattnig and Kerner, 1980), SAMM (Lamb and al, 1978), HOS (Hamilton and Zeldin, 1983) and RSL (Alford, 1985). These methods help to analyze applications domains in order to build physical and logical views of systems. In order to achieve this goal requirement specification methods introduce graphical notations and high level specification languages. It is a need that the notations and languages used lay on a formal framework to allow a rigorous verification and validation of the specification (Pong and Tse, 1991).

Requirement elicitation involves the system end user whom often lack of knowledge about the artificial languages used for the specification. Requirement elicitation is mainly a knowledge representation process where the analyst takes the user wishes expressed in a language close to natural language and map these wishes in the specification language. The requirement elicitation process automation mainly comes up against the lack of formalism of natural language. Because of these difficulties, requirement analysis methods cited so far don't take in account the elicitation step and this must be done manually by the system designer.

This report presents a review of some work done for the automation of user requirements acquisition. The next section defines user's requirement acquisition methods and the two type of requirements: functional requirements and non functional requirements. Section 1.2, is a presentation of functional requirements acquisition methods and section 1.3 a presentation on the treatment of non functional requirements.

1.1 User's requirement acquisition

Requirement Engineering includes the detection of requirements (elicitation, capture, validation and verification) and the representation of *how* to achieve the functional and non-functional characteristics required, independently to any realization. Formal specification languages or knowledge representation languages may be used for requirements representation (Pohl, 1992).

This review concerns users' requirement acquisition methods. A user's requirement acquisition method deals with the first stage of requirements engineering, the requirements detection phase. A user's requirements method represents users needs in a way that do not need a knowledge of particular representation method to understand. It is not obvious to draw the boundary between what we call a user requirement acquisition method and other requirements engineering methods. In this review, the boundary is based on the input of the method. Methods included in the review use sentences expressed in natural language or in subset of natural language such as *scenarios* and interactive user-oriented interfaces to get their input.

Many of the requirements engineering tools and methods begin at the requirements representation level, in providing a language with a formal syntax associated with formal or semi-formal semantics, to express requirements without any ambiguïty. Although these languages allow a good support for requirements engineering, they suppose software developers to make assumptions about users' needs and then to make a translation by "hand".

In general, a user's requirements acquisition method produces an output represented in a representation language but provide an automatic or semi-automatic mapping between users' requirements and the representation language.

Requirements are classified into functional and non-functional requirements (Roman, 1985). The functional requirements of a system concern the nature of interaction between the system and its environment while non-functional requirements state constraints on possible solutions.

Functional requirements capture is to get *what* users' need the system to do. The functional requirement acquisition and representation process results on a *conceptual model* (Blazer and Goldman, 1979) which models the relevant internal states and behavior of both the system under design and its environment.

According to a taxonomy defined in (Roman, 1985), non-functional requirements includes interface constraints, performance constraints, operating constraints, life-cycle constraints, economic constraints and political constraints.

- Interface constraints defines the system's and its environment interaction.

- Performance constraints concern:

  - time and space bound constraints such as response time, workload, storage space, etc,

  - reliably constraints which include the availability of physical components and data integrity,
  - security constraints and
  - survivability.

- Operating Constraints relates to the system operation context (personal abilities, system environment, ...).

- Life-cycle constraints includes maintainability, enhanceability, portability, flexibility, reusability and other issues such as development time limitation, resource availability and methodological standards.

- Economic constraints include the development cost and long term cost.

- Political constraints deal with policy and legal issues.

These non-functional requirements are grouped by (Bowen, 1985) in consumer oriented and technical oriented categories. Consumer oriented non-functional requirements are observable by the consumer. These requirements deals with the system efficiency, interoperability and completeness while technical oriented non-functional requirements include anomaly management, completeness and functional scope are addressed at the system level.

1.2 Functional requirements acquisition methods

1.2.1 Overview

We characterized user's functional requirements method by their input language, the representation of the output produced and the use of knowledge and tools to guide the acquisition process.

Table 1.1 summarizes the characteristics of the users' functional requirements acquisition methods covered by this survey. These users' functional requirements acquisition methods are SBSG (Hsia and Yuang, 1988), WATSON (Kelly and Nonnenmann, 1991), the KAOS meta-model (Dardenne et al., 1993), ALECSI (Cauvet et al., 1991), the Analyst Assist (Loucopoulos and Champion, 1989), TAMS (Dardenne, 1993), the Requirement Apprentice (Reubenstein and Waters, 1989), ASAP (Anderson and Fickas, 1989), a method developed by (Tsalgatidou et al., 1990) and a method developed by (Krogstie and McBrien, 1991).

Users' requirements methods input language

That is how users' requirements are expressed at the beginning of the use of the method. Input representation used by the studied methods are natural language, scenarios, graphics and interactive interfaces.

- Analyst Assist input is made of the informal natural language description of requirements. The analyst must however highlight the important concepts in the input document.

Method name	**Input**	**Output**	**Domain Knowledge**	**Supporting Tools**
SBSG	Graphical Scenarios	PASCAL prototype		
WATSON	Scenarios	Deterministic FSM	X	Temporal logic, Theorem prover
KAOS Meta-model	Interactive	KAOS Meta-concepts instances	X	Temporal logic
ALECSI	Natural language, Graphics	Conceptual structures	X	Expert-system
TAMS	Scenarios	KAOS Meta-concepts instances	X	
Analyst Assist	Natural language, Interactive	JSD functional spec	X	Fact resolver
Requirement Apprentice	Interactive	Knowledge base	X	Cake reasoning system
J. Krogstie	ERL rules, Graphical diagrams	Database schema, TEQUEL rules		
ASAP	Goals, Operators, Conditions	Operators and Conditions	X	AI planner
A. Tsalgatidou	Rules	Petri-nets, Prolog		Logic system, Petri-net animation

Table 1.1: Characterization of user functional requirements methods

ALECSI also use natural languages as input. Sentences can be used to describe real facts, static or dynamic constraints or rules governing the application. A semantic interpretation based on the meaning of the sentences verbs and their grammatical structure is used to derive semantic structures from natural language input.

- Scenarios describe the system under specification reaction to external stimulus or to the changes of its environment. Scenario are a natural way to describe the behavior of reactive systems but according to (Dardenne et al., 1993), scenario are also used to describe other kind of systems as information systems.

  In general, a scenario includes four parts. A *precondition* stating conditions which may hold prior to the scenario execution, a *trigger even* which is the external even making the system to react, an *action* executed by the scenario and *postconditions* which hold after the scenario execution.

  Scenarios in TAMS may include in addition characteristics such as the scenario author, the positivity or negativity of the scenario, the scenario situation, an initial state and a justification of the relevance of the scenario.

  SBSG scenarios are graphical representation of what the screen must look like during terminal sessions. SBSG is intended for applications where man-machine interaction use screens for data-entries such as database applications.

ERL and Tsalgatidou method rules are close to scenario with additional degree of formality for conditions. An ERL rule have the general following structure:

WHEN <trigger condition> IF <condition> THEN <conclusion>

Constraints rules are derived from the above structure when specifying only the *conclusion* part, derivation rules when the *WHEN* part is absent and action rules is the case where all the parts of the rule are specified.

In the method developed by Tsalgatidou and al, dynamic rules have a similar syntax to ERL rules. ASAP use in addition, static rules to describe constraints on the static structure of systems.

- Some methods acquire user's requirements by using a questionnaire-like refinement process through an interactive interface.

  Analyst Assist uses a dialogue formulator to ask appropriate questions to the user in order to build conceptual networks for concepts that its do not know, included in the input sentences.

  In the KAOS meta-model, requirements for applications are acquired by an interactive interaction where general knowledge is used to infer the particular knowledge relative to the application.

The output produced

User's requirements acquisition methods map user's requirements to a conceptual model specified in a requirement representation language or build a running prototype which realizes users' needs.

SBSG produces a prototype in PASCAL that realize the screens animation specified by users.

Krogstie and al method produce TEQUEL rules and C code which may be executed using a temporal rule manager.

Two kind of requirement representation languages are used as output. Formal specification languages and knowledge representation languages such as logic, semantic networks, frames and conceptual networks. Formal specification are produced by WATSON, and Tsalgatidou and al method. WATSON produces a set of Finite State Machines representing the features described by scenarios while Tsalgatidou and al method derives petri-net descriptions.

ALECSI produces conceptual structures representing the natural language sentences parsing. The KAOS meta-concepts is also a kind of conceptual structure that is enriched with Object-Oriented concepts. Instances of the KAOS meta-concepts are produced by TAMS and the KAOS Metamodel requirement acquisition system.

Knowledge usage in user's requirements acquisition

Requirement acquisition is an intensive cognitive task (Blazer and Goldman, 1979), where users needs are gotten and mapped in a conceptual model. This task requires intelligence and a great expertise. Many of the automatic methods for users' requirements acquisition use artificial intelligence technics for the purpose of representing knowledge about specific applications and the general background needed to understand the application domain. Knowledge representation is also used for reasoning in order to infer complete knowledge from partial one and to verify users requirements against general constraints. Due to the lack of formalism of natural language, methods that use this form of input relies on knowledge bases where the general information about the domain is stored. That is the case for ALECSI and the Analyst Assist.

WATSON use an huge knowledge base to represent all the knowledge about the application domain. These information are especially used to correct routine omissions and errors in scenarios, constraint the space of possible scenario generalizations, shape the structure of models used in model-based reasoning, and plan queries posed to the human designer.

The KAOS Meta-concept is by definition an abstract representation of the concepts of the world independently to any application. Requirements for particular applications are only instantiations of this meta-knowledge.

Analyst Assist use three separate knowledge bases. A domain knowledge which hold general knowledge about the application domain, the user fact base which gather information given by users during the requirement acquisition and a knowledge base about the JSD method.

The Requirement Apprentice use a "cliché" library containing knowledge about the domain. The cliché library contains information about the system, the environment and the needs.

ASAP a priori knowledge is its set operator, used to derive plans needed to achieve desired goals.

Supporting tools for the requirement acquisition method

Supporting tools includes theorem provers, graphic systems and logic inference systems. These tools are used to help the acquisition and verification of users' requirements.

The most frequently used tool is a logic system. Temporal logic is used by WATSON and the KAOS meta-model requirement acquisition system. In WATSON temporal logic is used as an intermediary representation between scenarios and finite state machine representations. WATSON's application domain knowledge is also represented in temporal logic. A Theorem prover is used for the scenario verification in WATSON. The method used is essentially to prove facts asserted by translating scenarios in temporal logic against the domain knowledge.

Logic is also used in the method of Tsalgatidou and al to prove the correctness of systems static structure against the static rules.

The Requirements Apprentice uses the Cake system, which allows reasoning about equalities, propositional deduction, maintenance of dependencies between deduced facts and incremental retraction of previously asserted facts. Cake is used at the heart of RA for the requirements acquisition and the building of the knowledge base.

ASAP use a Planner to produced a set of operators and initial operators to be used to reach a desired solution.

ALECSI have an expert-system architecture for the building and verification of system's specification base.

1.2.2 Methods

(Dardenne et al., 1993) present the KAOS meta-model, an approach for requirement acquisition. The KAOS meta-model allows to acquire requirements referring to the entire composite system including the part to be automated, its physical environment, and the way both parts have to cooperate.

The KAOS meta-model is a conceptual model. A graph where each node captures an abstraction and each edge a semantic link. Abstractions used by the meta-model are goals, actions, agents, entities and events. The elements of the meta-model are application-independent knowledge defined by features. Entities, relationships and events are described using an acquisition language to describes their features and by defining invariant conditions on them. An action is a mathematical relationship over objects. Actions attributes are preconditions, trigger conditions and postconditions. These attributes are asserted using temporal logic. Agents are object processors for some actions. Agents are able to observe the state of other objects known in the model.

Requirement acquisition in the KAOS meta-model is done in a learning by instruction framework. Requirements about the composite system are acquired as domain specific instances of elements of the conceptual meta-model. KAOS meta-model allows to capture objectives of the system, constraints that make such objectives operational, agents that control the system's behavior according to such constraints, events that cause the application of actions on entities, etc. The acquisition process is guided by strategies and domain models (specific ways of traversing the meta-model graph). A strategy is composed of question answering, input validation against meta-constraints, application of tactics to select preferred alternatives, deductive inferring or analogical reuse of domain models. The different kind of strategies are goal-directed, view-directed and scenario-directed strategies.

The result of requirement acquisition is expressed in an acquisition language. Its consist on domain specific instances of KAOS meta-concepts, linked by instances of meta-relationship and characterized by instances of meta-attributes. Formal checking of requirement is possible by the use of meta-level constraints and rules of inference based on temporal logic.

(Dardenne, 1993) presents a method that use *scenarios* for requirements acquisition. According to a study, the author established that scenarios are a natural way for users to describe what they want a system to do. Scenarios are characterized according to their length, positive/negative nature, automation degree, initial state, justification of relevance and degree of genericity. These characteristics conduct to the definition of a tool (TAMS) for system description that will be implemented in the future.

TAMS (Tool for Acquiring and Merging Scenarios) aims at scenarios acquisition using a language defined for scenarios description and the merging of several scenarios avoiding conflicts which may exist between them.

TAMS use domain knowledge and represents scenarios actions and predicates in an extension of the KAOS meta-model (Dardenne et al., 1993).

The Screen-Based Scenario Generator (SBSG) (Hsia and Yuang, 1988) is an interactive tool used to create, edit, compile and simulate systems model without a need to know any specific language. SBSG is motivated by the desire to be able to make rapid prototyping without the need of the knowledge of specialized languages, and then to make rapid prototyping tools accessible to non-professional users.

A scenario is defined by SBSG as a sequence of human and machine interaction in order to achieve users requirements. Scenario prototyping allow to get a system external behavior. In SBSG, scenarios clearly describes computers screens during interactive session. To each screen is associated a screen logic which details the screen fields and binds several screens each other by linking their fields.

SBSG includes tools for systems animation and a screen compiler which generates PASCAL programs from scenarios.

WATSON (Kelly and Nonnenmann, 1991) read and interprets informal natural language scenarios (examples of interaction) of telephone features, such as call waiting or call forwarding, and derives a plausible formal specification for a system that implement the described scenarios. WATSON also derives tests and refines the conjecture by posing extended behavior scenarios in natural language to a human developer.

WATSON use a large amount of domain knowledge encoded in temporal logic and automatic theorem proving to correct routine omissions and errors in scenarios, constraint the space of scenarios generalization. Domain knowledge and the context of the scenarios are also used to formulate questions to the designer in order to obtain additional information needed by WATSON.

Domain knowledge used by WATSON is specific to telephony application domain. Its include knowledge on hardware, network protocols and end-user etiquette. Using this domain knowledge as an a Priori information, WATSON do:

- Initial interpretation of the scenario by:
  - converting it from natural language into temporal logic term,
  - building a goal-directed plan (single stimulus-response cycle) (*episode*) for each scenario,
  - formulating for each *episode* a *transition rule* that is a temporal logic axiom guessing sufficient condition on the episode.

  This initial interpretation results in a rough finite-state design of the feature-control agents.

- Interactive refinement of the first design obtained to eliminate inconsistencies, nondeterminism, dead or unreachable section in agents design.
- Fault consistency test to assure that no temporal constraints of the domain knowledge is violated.
- Postprocessing to build executable prototypes, high-level code designs or to produce test data.

WATSON provides limited support for domain analysis through a tool kit for accessing and modifying WATSON'S formal theories of the telephone domain.

ALECSI (Cauvet et al., 1991) is a conceptual modelling expert system which support users requirements acquisition and validation for Information Systems. ALECSI takes user's requirements expressed either with natural language sentences or in graphical form (entities, actions, events and object-oriented view of the Information System) as input and produces conceptual structures. During this process, ALECSI supports two kind of tasks:

- knowledge engineering tasks which are the set of knowledge based tasks. Knowledge engineering tasks include the acquisition of the domain knowledge, the abstraction and conceptualization of the relevant parts of the application domain, the generation of the conceptual schema and the validation of the knowledge about the system.

- The second kind of tasks supported are process engineering tasks including guidance, explanation of the achieved results and interfacing with analysts and users.

ALECSI provides a set of graphical tools for the knowledge base building and scanning.

(Loucopoulos and Champion, 1989) presents a support environment for requirements engineering *Analyst Assist*. *Analyst Assist* assists during requirement elicitation, modelling and verification. The support environment use:

- A fact base where facts are represented like conceptual structures. These conceptual structures are built by using: a text analyzer which analyze keywords indicated by the user in the requirements texts, a concept editor used to add concepts in the knowledge base and a dialog interface.

  The requirement elicitation process is supported by a domain knowledge base which contains predefined conceptual structures representing contextual knowledge.

- A fact resolver that allow incremental construction of the base and

- A tool for tracing analysis decisions.

The application of the method result in a knowledge base which models the applications domain. *Analyst Assist* uses prototyping animation in order to validate the specification. The knowledge base expressing user's requirements is subsequently translated in an functional specification represented in the JSD method.

RA (Requirement Apprentice) (Rich et al., 1987; Reubenstein and Waters, 1989) is an interactive tool used to get software requirements. RA assists designers to elaborate complete and formal requirement specifications from informal one. RA produces:

- interactive results on the requirements conclusions and inconsistencies,
- a knowledge base built from requirements and
- documents based on requirements which may be used for contracts.

The RA reasoning is supported by Cake (Rich, 1985) a knowledge system which allows reasoning on equalities and proportional deduction. Cake also includes facilities permitting the maintenance of the deducted facts dependencies and the incremental retract of the asserted facts. The domain knowledge used by RA is represented in a "cliche" library and may be expanded.

A Requirement analysis by RA is done in two steps:

1. Definition of the system objects. The set of objects so defined constitutes the basic environment.
2. Definition of the functional requirements. Requirements ambiguities are exhibited during this step. The knowledge base representing the requirements is also built during this phase.

All the interaction between the system analyst and the RA are interactive by commands.

(Krogstie and McBrien, 1991) presents an information system development method which integrates process and rule based approaches. The method allows to mix both static and dynamic aspects including temporal dimension. The method is based on incremental development where details are successively added until a specification from which executable code can be derived is obtained.

A system description start by the description of its conceptual model. Three descriptions are provided for this purpose:

- The Phenomenon Model an Extended Entity Relationship Model which describes the static aspects of the real world. This model use entities, objects, connections and data types as modelling constructs.

- The Process Model, an Extension of Data-flow diagrams used to specify processes and their interaction in a formal way (interactions at the same level of abstraction and decomposition relationship at different levels). The process model uses processes, stores, agents, flows, ports and timers. Its allows a top-down modelling going from high level abstract processes and refining them.

- The External Rule Language (ERL) used to describe the internal behavior of processes and the constraints associated to processes at any level of abstraction. ERL is based on first-order temporal logic. Processes behavior is described as a set of rules. ERL rules can be distinguished in:

  - Constraint rules which are conditions that must not be violated,
  - Derivation rules which are rules used to derived information from other information present in the system and
  - Action which may be specified by a trigger condition, a condition and the derived conclusion.

ERL rules can be translated in Temporal rule manager (TEQUEL) form, and thus becomes directly executables.

After the conceptual model description, database schemas are automatically generated from the phenomenon model, complete and refined ERL rules description are derived from the process structure and logic, TEQUEL rules are then generated from the ERL rules and the system prototype may be executed.

(Anderson and Fickas, 1989) presents ASAP a system which use planning for specification design.

Planning in Artificial Intelligence is a technic used to select a list of operators to apply in order to reach a goal from an initial state. The plan is build using the goal, the initial state and a set of operators.

The approach used by ASAP encodes experts knowledge as operators which use the domain objects and relations, and build plans with this set of operators. Differently to AI planning, not only one but several initial states may exist and it is a need to avoid undesirable situations. ASAP generates a plan realizing users goals and exclude faults by generating plans to reach excluded conditions.

The problem description includes:

- goals which consists on positive goals and negative one,
- operators including:
  - environmental operators (these operator are independent to the

  designer),

  - artifact operators (user operators),

  - beginning operators (very expensive intended for a few use)

- initial conditions

ASAP produces in result a set of operators and initial conditions.

During the specification process, a trial set of operator and initial conditions are generated, this specification is analyses according to faults. The analysis proceed by generating prohibited plans. When the generated specification does not satisfy the goals without faults, conditions and operators are modified and the design process is resumed.

(Tsalgatidou et al., 1990) authors' approach investigates how user-oriented formalisms and techniques could be employed for specifying and capturing requirements. They propose the use of rules as a natural means for expressing the application domain knowledge, and introduce a number of techniques such as semantic prototyping and animation for the validation of the requirements.

The method presented is particularly intended for the specification and capture of requirements of domain intensive, transaction-oriented applications. The method lies on the assumption that requirements for such systems can be captured in term of rules conveying information about various aspects of the structure and the behavior of the domains.

The method proposes that a system description consist on static modelling constructs and dynamic modelling constructs.

Static modelling constructs is based on an extension of the entity relationship model with the addition of static rules to entities and relationships. Static rules are used for specifying domain knowledge which cannot be expressed by entities and relationships alone. A static rule is a linguistic expression which describes the state of affairs in the application domain at any time.

Dynamic modelling constructs describe events occurring in the system as dynamic rules which consist on a trigger condition, an action and a precondition.

The method introduce a number of techniques for the validation of the requirements model. The static model validation proceed by the mapping of the model to an executable logic language which combines the advantages of a rigorous formalism with those of rapid prototyping. The internal checking involve checking the well-formedness of the model according to the syntax rules of the modelling formalism and checking the consistency and completeness of the rules for the detection of self-contradicting or inconsistent rules. The static model checking use *semantic protyping* a validation method in which the user have an active participation. The validation method tries to prove that the model admits an interpretation which is inconsistent and tries to find interpretations agreeable to the user.

The dynamic model validation is used to prove the internal consistency of the model's behavior, its consistency with respect to the user's perception of the system and the model completeness. Dynamic rules are converted in a petri-net representation and animation of the representation helps in detecting redundant situations, conflicting situations, circular rules and missing rules.

1.3 Non-functional requirements

Unlike functional requirements, non-functional requirements are difficult to formalize at user level. According to (Roman, 1985) that is due to the fact that some constraints (eg response to failure) are related to design solution not known at the requirements specification step, human factors requirements may be determined only after complex empirical evaluations. Many constraints such as maintenability cannot be formalized and other are not explicit.

This is why there is no method in the literature dealing with non functional requirements at the user level. Although the question of non-functional requirements treatment has been tackled in later stages of the Software Engineering process.

Two approaches for the treatment of non-functional requirements exist (Mylopoulos et al., 1992). *Product-oriented* and *process-oriented* methods.

- Product-oriented methods defines software quality metrics that the software system must meet. A Software quality metric is a function that provides a way to develop quantitative, testable software quality requirements (Keller et al., 1992). This function takes as input a software data and produce a single numerical value that is the degree to which the software possesses a given attribute. Metrics may be used to provide visibility on software characteristics, to estimate software parameters such as timing, number of errors, effort, or utilization of a resource, as a standard for acceptance of a software product or for the validation of the software product.

A software quality metric goal is a score (or conditions on a score) for software quality metric attribute that ones hope to achieve. Software quality requirement are conditions or set of conditions defined in terms of software quality metrics that must be met to satisfy requirements.

RADC (Bowen, 1985) is a methodology for specifying and evaluating software quality metrics. RADC defines metrics attributes and provides guidelines for organizing and relating metrics attributes.

- The Process-oriented approach presented in (Mylopoulos et al., 1992) rationalizes system development process in terms of non-functional requirements. Non functional requirements are used to justify design decisions during the development process instead of evaluating the final product.

  The design is represented as a goal graph structure where:

  - Goals represents non functional requirements, design decision and arguments relating to other goals. There are three kind of goals:

    1. non functional requirements goals corresponding to accuracy, security, development, costs, performance,
    2. satisficing goals which are different categories of decisions that might be adopted in order to satisfy non functional require-

ments, and

3. argumentation goals representing formally or informally stated evidence or counter-evidence for other goals or goal refinements

– Link types relate goals or goals relationships to other goals. Links relate parents goals to their offsprings or indicates that an argument offers positive or negative support for a goal refinement by linking to argumentation goals.

The framework also includes:

– generic methods for refining goals into other goals,

– correlation rules used to infer potential interaction among goals and

– a labelling procedure that determine the degree to which non functional requirements are addressed by design decisions. The labelling procedure use propagation rules that allow for a semi-automatic propagation of nodes values to sub-nodes.

This framework has been specialized for dealing with accuracy requirements, performance requirements (Nixon, 1993) and security requirements (Chung, 1993).

Part 2

Survey On Incremental Specification methods

This second part is a presentation on incremental specification methods. The first section provides an overview of incremental specification methods. In this section we define incremental specification, its usage and the different kind of methods founded in the litterature. Section 2.2 summarize a selection of incremental specification methods.

2.1 Incremental Specification methods Overview

Incremental specification is a specification building method where elements are added piece by piece in order to obtain a full definition.

An incremental specification method allow to deal with incomplete description and provide a way to refine these incomplete specification until a complete specification is obtained. The important characteristic of incremental development methods is that incomplete specifications may be treated like complete specifications and the addition of a piece of specification do not oblige to treat the specification from the begining.

Incremental specification may be useful in the following phases of software engineering, requirement analysis, system design and maintenance.

- In the requirement analysis phase incremental methods are well suited for dealing with users' requirements that are often incomplete (Clerici and Oreja, 1990). Provided with incremental method for requirement analysis, the designer would be spare of making assumptions in order to complete users' requirements.

- During the design phase, systems are generaly decomposed vertically in several level of abstraction and horizontally in components. Incremental design method may allow system development abstraction level by abstraction level when providing a way to refine abstract definition in more concrete one like in (Terwilliger and Campbell, 1989).

  The horizontal development of systems may also be handled when the incremental method allow the merging of specifications like in (Khendek and Bochmann, 1993a) and (Ichikawa et al., 1990).

- During the system maintenance phase incremental development method allow the safe modification of systems. Maintenance is generally motivated by users' requirements modification. Users may want additional feature to be added or a modification or deletion of existing feature.

> Incremental development method that allow a modification of existing specification is very useful in such situation because the maintenance may be done without having to alter components not needed to be altered.

A great part of the work done for incremental specification concerns the extension of existing specification languages to support incremental specification development.

(Ichikawa et al., 1990), (Khendek and Bochmann, 1993a) and (Lloret et al., 1990) are all methods applied or applicable to the ISO specification language LOTOS (ISO8807, 1987). (Ichikawa et al., 1990) and (Lloret et al., 1990) introduce additional operator for merging specifications, while (Khendek and Bochmann, 1993a) presents an algorithm that take two specifications and produce a specification whose behavior is the merging of the behaviors of the two given specifications.

Incremental specification in the CCITT specification language SDL (X/3, 1992) is considered in specification development environments such as SDE (Ichikawa et al., 1991) and ESCORT (Wakahara et al., 1989). These environment use Message Sequence Charts as a high level description language for SDL specifications and achieve incremental specification by the ability to build an SDL complete specification by the addition of partial Message Sequence Chart specifications.

A method for incremental specification in Z (Spivey, 1989) is studied in (de Vasconcelos and McDermid, 1992). The method idea is to allow the modification of Z specifications without having to check the whole specification after every modification but only the part affected by the modification.

The other approaches presented GSBL (Clerici and Oreja, 1990) and ENCOMPASS (Terwilliger and Campbell, 1989) introduce new languages that allow incremental specification.

2.2 Incremental specification methods

Ichikawa and al present a method for incremental specification in LOTOS (Ichikawa et al., 1990).

Incremental specification is defined as the creation of a process B_{new} from a process B_{old} such that B_{new} **ext** B_{old}. With the process extension **ext** defined in (Brinksma et al., 1987):

The method extends LOTOS with an additional operator $\oplus$ called *merge* that have the following rules:

$$\begin{array}{lcl} B1 - \mu \rightarrow B1', B2 - \mu \rightarrow B2' & \vdash & B1 \oplus B2 - \mu \rightarrow B1' \oplus B2' \\ B1 - \mu \rightarrow B1', B2 \not- \mu \rightarrow B2' & \vdash & B1 \oplus B2 - \mu \rightarrow B1' \\ B1 \not- \mu \rightarrow B1', B2 - \mu \rightarrow B2' & \vdash & B1 \oplus B2 - \mu \rightarrow B2' \end{array}$$

It is shown that for every B1, B2. B1 $\oplus$ B2 **ext** B1 and B1 $\oplus$ B2 **ext** B2. Thus using $\oplus$, a behavior specification B_{new} may be incrementally constructed from a specification B_{old} by merging B_{old} with additional behavior B_{added}.

The incremental specification method supports system development in architectural context but do not consider internal events.

(Khendek and Bochmann, 1993b) presents a method for merging monolithic specifications. B1 and B2 being behavior descriptions, Merge(B1,B2) result on a new behavior description B such that B may exhibits behaviors of B1 and behaviors of B2 in an exclusive manner.

The combination of B1 and B2 in B is such that after doing a minimal cyclic trace of any of B1 or B2, B may exhibit, without any failure behavior in B1 or B2. A minimal cyclic trace being defined as a sequence of actions that start at the initial state and reach the initial state such that the does not contain any other cyclic trace.

It is proved that Merge(B1,B2) is such that the minimal cyclic traces in B1 and the minimal cyclic traces in B2 remain minimal in Merge(B1,B2) iff a certain condition holds.

The Merge algorithm has been extended in (Khendek and Bochmann, 1993a) for the incremental specification of structured systems.

The approach suppose two structured specifications S_{old} and S_{added} and construct a new specification S_{new} that have the same internal structure as S_{old} when a certain condition holds.

A structured specification is composed of components that may be structured components (and then composed themselves of other components) or basic end components. The algorithm for structured merging is recursively called for merging components until basic components are reached and are merged using the merge algorithm defined in (Khendek and Bochmann, 1993b). The algorithm assume that S_{old} and S_{added} have the same internal structure. A procedure that transforms structured specification that are different in order to obtain bisimilar specifications may be used for specifications that do not have the same structure.

The approach has been developed for behavior modelled as acceptance graph or labelled transition systems.Therefore the method can be applied to languages such as LOTOS or CSP since labelled transition systems are the underlying semantic model for these languages.

ENCOMPASS (Terwilliger and Campbell, 1989) is an environment for software development where implementation are obtained from requirement specification by doing incremental refinement.

Software configuration is described using an entity/relationship model enriched with aggregation and software requirements are specified in a language called PLEASE (Terwilliger, 1989).

PLEASE is an executable specification language that support program development by incremental refinement. PLEASE is based on logic and allow to specify functional requirements by using preconditions on input values and postconditions on the result. These conditions are specified as predicates.

ENCOMPASS is composed of:

- ISLET a language oriented editor for the construction and the refinement of PLEASE specifications,
- TED a proof management system that use theorem provers,
- a prototyping tool and
- a test harness.

ISLET is used to create PLEASE specifications and incrementally refine them into ADA. Verification conditions may be generated during the refinement process. These verification conditions are proved using TED. Valid specification are used to generate executable prototypes in PROLOG and ADA. The test harness is used by user to select test case. Once a specification is completed at a certain level, it may be further refined and then start another validation process. That is done until a final implementation is obtained.

GSBL (Clerici and Oreja, 1990) is an algebraic specification language intended to serve as support for the incremental development of specifications from requirements.

GSBL is built over the following principles:

- the possibility to deal with incomplete specifications,
- genericity,
- inheritance and
- a powerful binding mechanisms that take in account specification structure.

In GSBL, a class is a specification unit. Class consists on sorts, operations and equations that may be partially defined.

A new class may be defined from existing one by extension or a consistent refinement of an incompletely defined class. Incremental development may be done by using such refinements from an incomplete specification to a more complete and finalized one.

A method for merging Labelled Petri-Net specifications is presented in (Lloret et al., 1990). Labelled Petri-Net behavior is introduced as a Labelled Transition System and the composition of two Labelled Petri-Net corresponds to the composition of the Labelled Transition System modelling their behavior.

The Labelled Transition System composition operator $|_{LTS}$ is defined as follow considering $proc_i = (S_i,\ \tau_i,\ -t_i \rightarrow_{t_i \in \tau_i},\ S_{i,0},\ L_i)$, i=1,2 to represent Labelled Transition Systems,

$proc_1\ |_{LTS}\ proc_2$ is $(S_1|S_2,\ \tau_1|\tau_2,\ -t \rightarrow_{t \in \tau_1|\tau_2},\ S_{1,0}|S_{2,0},\ L_1|L_2)$ where:

$L_1|L_2$ is $proc_1|proc_2$ labelling defined by the set of gates $(\alpha_{proc_1} \cup \alpha_{proc_2})$ labelling function l defined on domain $\tau_1|\tau_2$.

Function l is defined by the following rules:

1. $\frac{s_1 - t_1 : l_1 \rightarrow s_1^{'}}{s_1|s_2 - t_1 : l_1 \rightarrow s_1^{'}|s_2}\ (sync_{\alpha\cap}(l_1) = \emptyset)$

2. $\frac{s_2 - t_2 : l_1 \rightarrow s_2^{'}}{s_1|s_2 - t_2 : l_2 \rightarrow s_1^{'}|s_2^{'}}\ (sync_{\alpha\cap}(l_2) = \emptyset)$

3. $\frac{s_1 - t_1 : l_1 \rightarrow s_1^{'}, s_2 - t_2 : l_2 \rightarrow s_2^{'}}{s_1|s_2 - t_1|t_2 : (l_1 \cup l_2) \rightarrow s_1^{'}|s_2^{'}}\ (sync_{\alpha\cap}(l_1) = sync_{\alpha\cap}(l_2))$

$sync_{\alpha\cap}(l_i)$ is the set of events of $proc_i$ whose gate are shared with events of the other process.
l_i is a labelling function that associate a transition in τ_i to an action.

Labelled petri net operator $|_{LPN}$ is defined such that $Pn_1|_{LPN}Pn_2$ corresponds to $LPn_1|_{LTS}LPn_2$ when LPn_1 express the behavior of Pn_1 and LPn_2 express the behavior of Pn_2.

(de Vasconcelos and McDermid, 1992) propose a method that use an hypertext like tool for the incremental processing of Z (Spivey, 1989) specifications.

The use of hypertext allow the linking of different versions of a specification, specification refinement, specifications relation, specification documentation and provide an abstraction view on specifications to users. Using the hypertext tool, the description of systems is introduced, modified and browsed interactively.

The method proposes an incremental checking algorithm that check only the part of a specification needed to be checked after its modification, because editing a definition may invalidate the type environment under which other definitions were type-checked. The essence of the incremental type-checking is to reduce the retypecheching of a specification to the minimum necessary when it is edited. It is only needed to check the dependent definitions affected by the editing.

The basis of the proposed algorithm is the linking between interdependent definitions that allow to trace all propagation of the modification of specifications.

Incremental development is supported in the sens that the hypertext tool may be used to edit definitions that are more rapidly checked by the incremental type-checking algorithm.

The System Design Environment SDE (Ichikawa et al., 1991) is intended for the incremental specification of communication systems. SDE supports system description, system validation, design document generation, program generation and system maintenance.

A communication system is described as a set of service specification using a textual language SAL (Ichikawa et al., 1986) or a graphical language GSAL (Ichikawa et al., 1988). SAL and GSAL are based on Message Sequence Charts and then allow to represent services as partial behaviors. The system is validated by checking the feasibility of services, detecting inconsistent services and un-required behavior. SDE generates SDL process specification from correct services description.

A process behavior generation is done incrementally by updating the process global behavior according to partial behavior described using SAL charts.

Program generation in SDE is assured by KINDRA (Dell et al., 1981) which transform SDL process description in process skeleton code in high level programming languages such as C, CHILL and PASCAL.

One of the major strength of SDE reside on its support of system evolution by the incremental feature of SDE which permits existing process behavior modification according to new services.

Conclusion

The project M.c.NSERC will use Scenarios as a basis for users' requirements expression. The ultimate goal of the project is to build a development tool that will allow designers in collaboration with users to design systems using scenarios as a departure. This project involves dealing with concepts from various research domains such as users' requirements engineering, incremental specification, requirement tracability and real-time system design.

This survey has presented the recent results on users' requirements engineering and incremental specification. A number of learnings drawn from this study will be take in account during the subsequent phases of the project.

Bibliography

Alford, M. W. (1985). SREM at the age of eight: the distributed computing design system. *IEEE Computer*, 18(4):36–46.

Anderson, J. S. and Fickas, S. (1989). A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem. *ACM SIGSOFT Software Engineering notes*, 14(3):177–184.

Blazer, R. and Goldman, N. (1979). Principles of Good Software Specification and Their Implications for Specification Languages. In *Proceedings of Specification Reliable Software Conference*, pages 58–67.

Boehm, B. W. (1976). Software Engineering. *IEEE Transaction on Computers*, C-25(12):1226–1241.

Boehm, B. W. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, 1(1):75–88.

Bowen, T. P. (1985). Specification of Software Quality Attributes. Technical Report RADC-TR-85-37, Rome Air Development Center.

Brinksma, E., Scollo, G., and Steenbergen, C. (1987). LOTOS Specifications, Their Implementations and Their Tests. In *Protocol Specification, Testing, and Verification, VI*, pages 349–360. North-Holland.

Cauvet, C., Proix, C., and Rolland, C. (1991). ALECSI: an expert system for requirements engineering. In Andersen, R., Bubenko, J. A., and Solvberg, A., editors, *Advanced Information Systems Engineering. Third International Conference CAiSE '91 Proceedings*, pages 31–49. Springer-Verlag.

Chung, L. (1993). Dealing with Security Requirements during the Development of Information Systems. In Roland, C., Bodard, F., and Cauvet, C., editors, *Advanced Information Systems Engineering, Proceedings, Fifth International Conference CAiSE'93*, pages 234–251. Springler Verlag.

Clerici, S. and Oreja, F. (1990). The specification language GSBL. In *Recent Trends in Data Type Specification. 7th Workshop on Specification of Abstract Data Types. Proceedings*, pages 31–52. Springer-Verlag.

Dardenne, A. (1993). On the Use of Scenarios in Requirements Acquisition. Technical Report CIS-TR-93-17, Department of Computer and Information Science University of Oregon.

Dardenne, A., van Lamsweerde, A., and Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1-2):3–50.

de Vasconcelos, A. and McDermid, J. A. (1992). Incremental processing of Z specifications. *IFIP Transactions C [Communication Systems]*, C-10:53–69. Fifth International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols - FORTE '92. IFIP TC6/WG6.1 Conf. Date: 13-16 Oct. 1992.

Dell, P. W., Jackson, L. A., O'Brien, R., and Tinker, R. (1981). Computer aided design for software. *Software & Microsyst.*, 1(1).

Dickover, M. E., McGowan, C., and Ross, D. T. (1978). Software design using SADT. *Structured Analysis and Design*, 2.

Hamilton, M. and Zeldin, S. (1983). The functional life cycle model and its automation: USE.IT. *Journal of Systems and Software Science*, 3(3):25–62.

Hsia, P. and Yuang, T. A. (1988). Screen-based Scenario generator A tool for scenario-based prototyping. In *Software Track Proceeding of the twenty-first Annual Hawaii International Conference*, pages 455–461. IEEE Computer Society Washington DC.

Ichikawa, H., Itoh, M., Kato, J., Takura, A., and Shibasaki, M. (1988). A Graphic Language for Telecommunication Services Based on the Message Sequence Chart. *SIG Report on Software Engineering*, 63(2).

Ichikawa, H., Itoh, M., Kato, J., Takura, A., and Shibasaki, M. (1991). SDE: Incremental Specification and Development of Communications Software. *IEEE Transaction on Computers*, 40(4).

Ichikawa, H., Itoh, M., and Shibasaki, M. (1986). Protocol-Oriented Service Specification and their Transformation into CCITT Specification and Description Language. *Trans. IECE Japan*, E.69(4):524–535.

Ichikawa, H., Yamanaka, K., and Kato, J. (1990). Incremental specification in LOTOS. In *Protocol Specification Testing and Verification, X. Proceedings of the IFIP WG 6.1 Tenth International Symposium*, pages 183–196. North-Holland.

ISO8807 (1987). LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour.

Jackson, M. A. (1983). *System Development.* Prentice-Hall Series In Computer Science.

Keller, S. E., Kahn, L. G., and Panara, R. B. (1992). Specifying Software Quality Requirements with Metrics. In Thayer, R. H. and Dorfman, M., editors, *System and Software Requirements Engineering.* IEEE Computer Society Press Tutorial.

Kelly, V. and Nonnenmann, U. (1991). *Reducing the Complexity of Formal Specification Acquisition*, chapter 3, pages 41–64. AAAI Press/The MIT Press.

Khendek, F. and Bochmann, G. (1993a). Incremental Construction Approach for Distributed System Specification. In *FORTE'93.*

Khendek, F. and Bochmann, G. (1993b). Merging Behavior Specifications. Technical Report 856, Université de Montréal.

Krogstie, J. and McBrien, P. (1991). Information System Development Using a Combination of Process and Rule Based Approaches. In *Advanced Information Systems Engineering. Third International Conference CAiSE '91 Proceedings*, pages 319–335. Paris 1 Univ., France, Springer-Verlag.

Lamb, S. S. and al (1978). SAMM: a modeling tool for Requirements and Design Specification. In *Proceedings of the 2nd IEEE International Software and Applications Conference (Compsac '78)*, pages 48–53.

Lloret, J. C., Azema, P., and Vernadat, F. (1990). Compositional design and verification of communication protocols, using labelled Petri nets. In *Computer-Aided Verification. 2nd International Conference, CAV '90 Proceedings*, pages 96–105. Springer-Verlag.

Loucopoulos, P. and Champion, R. (1989). Knowledge-based support for requirements engineering. *Information and Software Technology*, 31(3):123–135.

Mylopoulos, J., Chung, L., and Nixon, B. (1992). Representing and Using Non-Functional Requirements: A Process-Oriented Approach. *IEEE Transaction on Software Engineering*, 18(6):483–497.

Nixon, B. A. (1993). Dealing with Performance Requirements During the Development of Information Systems. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 42–49. IEEE Computer Society Press.

Pohl, K. (1992). The Three Dimensions of Requirements Engineering. Technical report, NATURE project.

Pong, L. and Tse, T. (1991). An examination of requirements specification languages. *Computer Journal*, 34(2):143–152.

Pressman, R. S. (1992). *Software Engineering a Practitioner's Approach.* Mc Graw Hill, third edition.

Reubenstein, B. H. and Waters, R. C. (1989). The requirement apprentice: an initial scenario. *ACM SIGSOFT Software Engineering notes*, 14(3):211–218.

Rich, C. (1985). The Layered Architecture of a System for Reasoning about Programs. In *IJCAI85*, pages 540–546.

Rich, C., Reubenstein, B. H., and Waters, R. C. (1987). Toward a Requirements Apprentice. In *Proc. 4th International Workshop on Software Specification and Design*, pages 79–86. IEEE Computer Society Press.

Roman, G.-C. (1985). A Taxonomy of Current Issues in Requirements Engineering. *IEEE Computer*, pages 15–23.

Spivey, J. M. (1989). *The Z Notation: A Reference Manual.* Prentice-Hall.

Teichroew, D., Macasovic, P., Hershey-III, E., and Yamamoto, Y. (1980). Application of the entity-relationship approach to information processing systems modelling. In *Entity-Relationship Approach to Systems Analysis and Design*, pages 15–39. P.P.Chen.

Terwilliger, R. B. (1989). PLEASE: executable specifications for incremental software development. *Journal of Systems and Software*, 10(2):97–112.

Terwilliger, R. B. and Campbell, R. H. (1989). ENCOMPASS: an environment for the incremental development of software. *Journal of Systems and Software*, 10(1):41–53.

Trattnig, W. and Kerner, H. (1980). EDDA: a very-high-level programming and specification in the style of SADT. In *Proceedings of the 4th IEEE International Software and Applications Conference (Compsac '80)*, pages 436–443.

Tsalgatidou, A., Karakostas, V., and Loucopoulos, P. (1990). Rule-based requirements specification and validation. In Steinholtz, B., Solvberg, A., and Bergman, L., editors, *Advanced Information Systems Engineering. Second Nordic Conference CAiSE '90 Proceedings*, pages 251–263. Springer-Verlag.

Wakahara, Y., Kakuda, Y., Ito, A., and Utsunomiya, E. (1989). ESCORT: an environment for specifying communication requirements. *IEEE Software*, (3):38–43.

X/3, W. P. (1992). Recommendation Z.100 - The CCITT Specification and Description Language (SDL).