Amyot, D. and Eberlein, A. An Evaluation of Scenario Notations for Telecommunication Systems Development. In: Proc. 9 th ICTS (Dallas, TX, March 2001)

http://citeseer.nj.nec.com/amyot01evaluation.html

# An Evaluation of Scenario Notations for Telecommunication Systems Development

Daniel Amyot
Mitel Corporation, Kanata, Canada, and
School of Information Technology and Engineering, University of Ottawa
150 Louis-Pasteur, Ottawa, Ontario, K1N 6N5, Canada
e-mail: damyot@site.uottawa.ca
URL: http://www.site.uottawa.ca/~damyot/

Dr. Armin Eberlein
Department of Electrical & Computer Engineering, University of Calgary
2500 University Drive NW, Calgary, Alberta, T3A 5S5, Canada
e-mail: eberlein@enel.ucalgary.ca
URL: http://www.enel.ucalgary.ca/People/eberlein/

*Abstract*

The elicitation, modeling and analysis of requirements have consistently been one of the main challenges during the development of complex systems. Telecommunication systems belong to this category of systems due to the worldwide distribution and the heterogeneity of today's telecommunication networks. Many proposals have been made for approaches that allow the developer to describe system behavior. Scenarios and use cases represent one such approach that has become popular for the capturing and analysis of requirements. However, little research has been done that compares different approaches and assesses their suitability for the telecommunications domain. This paper defines evaluation criteria for scenario notations and then uses these criteria to review twelve scenario notations. In addition, nineteen approaches for the construction of more detailed and integrated design models from scenarios are briefly compared. Such models enable further analysis and pave the way towards automated generation of implementations. Hopes and challenges related to scenario-based development of telecommunication systems are finally discussed.

*Keywords*

Design, Model, Requirements, Scenario, Synthesis, Telecommunications, Use Case

## 1 Introduction

The modeling of telecommunication systems requires an early emphasis on non-functional requirements, followed by behavioral aspects such as interactions between the system and the external world and users, on the cause-to-effect relationships among these interactions, and on intermediate activities performed by the system. Over the years, several approaches have been used to provide notations for describing behavioral aspects of emerging telecommunication systems and services. On one hand, proponents of formal methods have claimed to solve the problem by providing unambiguous and mathematical notations and verification techniques, but the penetration of these methods in industry and in standardization bodies (especially in North-America) remains, unfortunately, low [5][12][34]. On the other hand, scenario-driven approaches, although often less formal, have raised a higher level of interest and acceptance, mostly because of their intuitive representation of services [37][38][48][78]. Such semi-formal notations are a good compromise between informal and formal approaches: they are more precise than natural language and more convivial than formal languages. Their application to requirements and the early stages of the design process raises new hopes for the availability of concise, descriptive, maintainable, and consistent documents, standards, and design specifications that need to be understood by a variety of readers. Moreover, scenarios pave the way towards the construction of detailed (formal) models and implementations through analytic and synthetic approaches. These construction approaches promise to generate models and implementations faster and at a lower cost while improving their correctness and traceability with respect to the requirements.

Because of an increasing interest in the very active field of scenario techniques and because there is a lack of comprehensive surveys for the telecommunication domain, this paper presents an evaluation of state-of-the-art notations and techniques for the scenario-based development of telecommunication systems and services. Section 2 uses eight criteria to compare twelve scenario notations applicable to the telecommunications domain. Section 3 presents an evaluation of nineteen analytic and synthetic approaches for the construction of implementation-oriented models from scenarios. Section 4 discusses challenges and hopes for the future, with a special emphasis on ITU-T's upcoming User Requirements Notation (URN), and Section 5 contains our conclusions. Due to the nature of this work, an extensive bibliography is included (Section 6).

# 2 Scenario Notations

## 2.1 Why Scenarios?

Scenarios are known to help describing functional requirements, uncovering hidden requirements and trade-offs, as well as validating and verifying requirements. The introduction of *use cases* in the object-oriented world confirmed this trend almost a decade ago [47]. Scenarios are used not only to elicit requirements and produce specifications, but also to drive the design, the testing, the overall validation, and the evolution of the system.

The exact definition of a scenario may vary depending on used semantics and notations, but most definitions include the notion of a *partial* description of system usage as seen by its users or by related systems [65]. There is no clear separation between the meanings of use case and scenario. In UML, use cases are defined as sequences of actions a system performs that yield observable results of value to a particular user (actor) [61]. In the object-oriented community, use cases are interpreted as classes of related scenarios, where scenarios are sequential and where use case parameters are instantiated with concrete values. Hence, a scenario is a specific realization of a use case [61][64]. However, the requirements engineering community sometimes sees multiple use cases as being contained in a scenario. In this paper, the terms “use cases” and “scenarios” are used interchangeably.

One frequent problem requirements engineers and designers are faced with is that stakeholders may have difficulties expressing goals and requirements in an abstract way [54]. Typical usage scenarios for a hypothetical system may be easier to obtain than goals or properties when the system understanding is in its infancy. This fact has been recognized in cognitive studies on human problem solving [15] and in research on inquiry-based requirements engineering [62].

The use of scenarios for requirement engineering and system design bears benefits and drawbacks. A non-exhaustive list of the most relevant ones follows in Table 1.

The increasing popularity of scenarios suggests that their benefits outweigh their drawbacks. Further, using scenarios in conjunction with other techniques can often cure these drawbacks.

| *Benefits* | *Drawbacks* |
|---|---|
| • Scenarios are intuitive and relate closely to the requirements. Different stakeholders, such as designers and users, can understand them. They are particularly well suited for operational descriptions of reactive systems and telecommunication systems.<br>• They can be introduced in iterative and incremental design processes.<br>• They can abstract from the underlying system structure, if necessary.<br>• They are most useful for documentation and communication.<br>• They can guide the requirements-based tests generation for validation at different levels (specification, design and implementation).<br>• They can guide the construction of more detailed models and implementations. | • Since scenarios are partial representations, completeness and consistency of a set of scenarios are difficult to assess, especially when the scenarios are not described at a uniform abstraction level.<br>• Scenarios are not able to express most non-functional requirements.<br>• Scenarios often leave required properties about the intended system implicit.<br>• The synthesis of components behavior, from a collection of scenarios, remains a complex problem.<br>• The use of scenarios leads to the usual problems related to traceability with other models used in the development process.<br>• Getting and maintaining the right granularity for the scenarios can be a real challenge<br>• Design approaches based on scenarios are rather recent and seldom possess a high level of maturity. Scalability and maintainability represent notably important issues. |

**Table 1** Benefits and Drawbacks of Scenarios

### 2.2 Evaluation Criteria

The following collection of eight important criteria will help to categorize and compare many scenario notations relevant to the development of telecommunication systems and services:

- **Component-centered**: Scenarios can be described in terms of communication events between system components only (i.e. *component-centered*), or else independently from components, in a pure functional style (i.e. *end-to-end*). This is a very important criterion as many notations focus solely on interactions between components, while in our view these interactions often belong to detailed design. An early focus on messages may lead to system overspecification and may prune out other appropriate options.
- **Hiding**: Scenarios could describe system behavior with respect to their environment only (black-box), or it could include internal (hidden) information as well (gray-box). According to Chandrasekaran [21], the most important reason that impeded the progress of various large projects he studied is the lack of internal details in scenarios. Essentially, treating the system like a black box in a scenario model means that there shall be no consideration of implementation constraints while describing scenarios. It does not mean that a scenario shall not delve into details of requirements on internal system functionality. Zave and Jackson present a different viewpoint and claim that when it comes to requirements, the environment is not the most important thing: it is the only thing [81]. They suggest avoiding any implementation bias on the basis that requirements are supposed to describe what is observable at the interface between the environment and the system, and nothing else about the system. Our opinion is more in line with Chandrasekaran's: shared events, whether they are controlled by the system or by the environment, are insufficient. Many implementation constraints are not necessarily premature design decisions, but in fact non-functional requirements. Additionally, there comes a point where the gap between requirements and high-level designs or implementations needs to be filled, and descriptions of activities performed internally by the system can then be of tremendous help.
- **Representation**: Scenarios can be described in various ways, for instance with semi-formal pictures, natural language, structured text, grammars, trees, state machines, tables, visual paths, and sequence diagrams. Graphical representations are often better understood by a wide range of stakeholders,

whereas structured textual languages are often less constrained in terms of expressiveness. The level of formality has also an impact on the usefulness of a notation: less formality is better for requirements, but more formality is desirable for detailed design and automated model transformations or code generation.

- **Ordering**: Scenarios represent a collection of events ordered according to *time* only or to *causality*. Causal ordering is very important when concurrency is involved (as it is the case in most telecommunication systems), otherwise concurrent actions expressed with a time ordering might result in logical fallacies and other incorrect assumptions at the requirements level.
- **Multiplicity**: We can either have one *single* trace only (i.e. a sequential scenario) or possibly *multiple* related traces per scenario. Having multiple scenarios linked together leads to more concise descriptions and to a better understanding of the integration of scenarios, whereas the availability of individual scenarios eases the construction of traceable links across design models. But obviously, a notation that can support multiple scenarios can support single scenarios as well.
- **Abstraction**: An *abstract* scenario is generic, with formal parameters, whereas a *concrete* scenario focuses on one specific instance, with concrete values. Abstraction is beneficial in the early stages of design (e.g. requirements capture) and for capturing families of scenarios that differ only by their concrete values. Notations that focus on concrete scenarios however ease the transition towards detailed models (e.g. state machines), test cases, and implementations.
- **Identity**: Scenarios can focus on *one actor* or target *many actors* at once. The later is seen as a major benefit in terms of expressiveness.
- **Dynamicity**: A scenario notation is *dynamic* when it enables the description of behavior that modifies itself at run-time, otherwise it is said to be *static*. Emerging telecommunication services enabled by IP networks, agent systems, and negotiation mechanisms can benefit from notations that can express dynamicity.

Obviously, other sets of criteria could be defined. For instance, Cockburn uses four dimensions to use case descriptions, namely purpose, content, plurality, and structure [23]. Purpose can be either for stories (explanations) or for requirements. Content can be either contradicting, consistent prose, or formal content. Plurality is either 1 or multiple, similar to our multiplicity. Structure can be unstructured, semi-formal, or formal. This dimension shares some common characteristics with our representation criterion. Rolland *et al.* suggest yet another set of criteria in [66], but without a real emphasis on specific needs of telecommunication systems.

The next section does not attempt to provide a single scenario definition. Instead, it presents and compares different notations according to the selected criteria.

### 2.3 Selected Notations

There are dozens of scenario notations used for the description of system usage, goals, and business logic. Hurlbut's thesis surveyed and compared nearly sixty different scenario, use case, and policy formalisms and models [37][38], and others are likely to emerge in the upcoming years. This section focuses on a selection of twelve scenario notations particularly relevant to the telecommunications domain, and it provides a concise comparison in terms of the criteria seen in Section 2.2.

- **Message Sequence Charts (MSCs)**. The scenario notation that is the most commonly used by telecommunications companies and standards bodies is undoubtedly Message Sequence Charts [45]. This notation describes exchanges of messages (arrows) between communicating entities (vertical lines). MSCs are essentially graphical (although a textual machine-processable format exists), composed of concrete events (messages), and centered towards components. MSCs can represent internal actions and multiple actors. While conventional MSCs mostly use time ordering and single traces (MSC'2000 now enables multiple traces), High-Level MSCs (HMSCs) focus on multiple structured scenarios and also on causality. MSCs have been used by many people to formalize scenarios. Kimbler *et al.* use them to create Service Usage Models, which describe the dynamic behavior of system services from the user's perspective [50][64]. Andersson and Bergstrand also present a method to formalize use cases that introduces an unambiguous syntax via MSCs [11].

- **Use Cases**. Jacobson's use cases are prose descriptions of behavior from the user's perspective [47]. They are mostly black-box, i.e. they focus on the interactions between actors and systems. UML use case diagrams offer a graphical means by which use cases can be related to each other [61]. They offer relations such as *uses* and *extends*, which allow for uses cases to reuse (part of) other scenarios. Use cases can be of two kinds: basic courses for normal scenarios, and alternative courses, which include fault-handling scenarios. Use cases are mostly based on time ordering, they represent multiple abstract scenarios, and they may involve many actors.
- **CREWS-L'Ecritoire**. CREWS, the European ESPRIT project on Cooperative Requirements Engineering With Scenarios, proposes structured narrative text for capturing requirements scenarios, together with a set of style and content guidelines [14]. This notation is supported by a tool called *L'Ecritoire* [66] and, to some extent, by the SAVRE tool [58]. In a way similar to Jacobson's use cases, the textual scenarios are divided into two main categories: normal scenarios and extension scenarios. The latter can be either normal (alternatives) or exceptional, depending on whether they allow to reach the associated goal or not. This notation supports multiple actors and abstract scenarios, focuses on external events, is centered towards components, and uses time ordering.
- **Scenario Trees**. Hsia *et al.* suggest the use of scenario trees that represent all scenarios for a particular user [36]. Similarly to Labeled Transition Systems (LTSs), scenario trees are composed of nodes, which capture system states, and of arcs representing events that allow the transition from one state to the next. They also focus on interactions between actors and the system, they use time ordering, and they can be abstract. This notation is best suited for a single thread of control and well-defined state transition sequences that have few alternative courses of action and no concurrency, which is seldom the case in real telecommunications systems. Regular expressions are used to formally express the user scenario that results in a deterministic finite state machine.
- **Use Case Trees (UCTs)**. Boni Bangari proposes Use Case Trees as a text-based notation for describing scenarios related to one entity [16]. This notation, inspired from the TTCN testing notation [40], captures sequential and alternative scenarios in terms of messages. These messages are sent and received through points of control and observations (PCOs) belonging to an actor under test. The grammar-like representation allows for sub-trees, timer events and data parameters (assignments, operations and qualifiers) to be defined and used. An interesting property of UCTs is that sequential scenarios can be generated automatically from the grammar (usually in the form of Message Sequence Charts) and characterized as normal, low risk, or high-risk scenarios. This notation is potentially useful for defining compact validation test suites targeted towards the system as a whole or towards single components. However, the lack of support for concurrency, multiple entities and hiding limits its usefulness as a requirements notation.
- **Chisel Diagrams**. Aho *et al.* have performed empirical studies with telecommunication engineers to create the Chisel notation [2]. The graphical language Chisel is used for defining requirements of telecommunication services. Chisel diagrams are trees whose branches represent sequences of (synchronous) events taking place on component interfaces. Nodes describe these events (multiple concurrent events can take place in one node) and arcs, which can be guarded by conditions, link the events in causal sequences. Multiple abstract scenarios and actors can be involved, but internal actions are not covered.
- **Statechart Diagrams**. Glintz uses Harel's Statechart notation [31], now part of the Unified Modeling Language (UML) [61], as a way of capturing scenarios [29]. This results in a formal notation for validating and simulating a behavioral model representing the external view of a system. Scenarios must be structured such that they are all disjoint. Any overlapping scenarios must be either merged into a single scenario or partitioned into several disjoint ones. Such structuring allows for each scenario to be modeled by a closed Statechart, i.e. a single initial state and a single terminal state, with other states in between. Composition of scenarios is performed though sequence, alternative, iteration, or concurrency declarations. These scenarios support causal ordering, multiple actors, and multiple abstract scenario sequences.
- **Life Sequence Charts (LSCs)**. Damm and Harel propose Life Sequence Charts [24], which enrich MSCs with a concept called *liveness*. Liveness enables one to specify mandatory scenarios as well as

forbidden scenarios (e.g. to capture safety requirements) through the same representation. Although the liveness concept is certainly useful and leads to more accurate component descriptions, LSCs satisfy essentially the same evaluation criteria as High-level MSCs.

- **Somé's Scenarios**. Somé *et al*. represent timed scenarios with structured text, but also with a formal interpretation where preconditions, triggers, sequence of actions, reactions and delays are specified [74][75]. Scenarios are interpreted as timed sequences of events, which make them appropriate for real-time systems. External events represent interactions between components, including actors, whereas actions can be internal. These textual scenarios can also be represented graphically. Somé extended the MSC notation to support additional scenario elements such as conditions and expiration delays (now covered to some extent by High-level MSCs). Multiple abstract scenarios and actors can be considered by these component-based notations. They are ordered according to time, although non-linear causality appears when composing the scenarios together to form an automaton.
- **RATS**. In the RATS (*Requirements Acquisition and specification for Telecommunication Services*) methodology [27], Eberlein uses three different scenario representations: textual (natural language), structured (in text, but with pre/post/flow conditions) and formalised (structured text, more component-centered). The aim of having these three notations is to allow a smooth and gradual transition from a service description in natural language to a formal specification in SDL. Scenarios are divided into normal, parallel/alternative, and exceptional behavior, in order to help the developer to first focus on the most common behavior and then later on the less common system functionality. The use cases can be structured hierarchically in overall use cases of higher abstraction. Most scenarios are abstract and linear, although overall scenarios capture multiple scenarios, with a causal ordering. The methodology has been implemented in a prototype of the RATS tool, which is a client-server-based expert system.
- **Use Case Maps (UCMs)**. The Use Case Map notation is used for describing *causal relationships* between *responsibilities*, which may potentially be bound to underlying organizational structures of abstract *components* [7][20][77]. Responsibilities are generic and can represent actions, activities, operations, tasks to perform, and so on. Components are also generic and can represent software entities (objects, processes, databases, servers, functional entities, network entities, etc.) as well as non-software entities (e.g. users, actors, processors), which can hide the responsibilities they contain. The relationships are said to be causal because they involve concurrency and partial orderings of activities and because they link causes (e.g., preconditions and triggering events) to effects (e.g. postconditions and resulting events). In a way, UCMs visually show multiple abstract and related use cases in a map-like diagram. Yet, UCMs do not specify message exchanges between components, which implement the causal flow of responsibilities. These messages are left to a more detailed stage of the design process. UCMs can also capture run-time self-modifying behavior through dynamic stubs and dynamic responsibilities [10].
- **UML Activity Diagrams** [61]. All UML behavioral diagrams can be used to describe scenarios. Four of them have already been discussed in some form in this section: Jacobson's use cases and use case diagrams, sequence diagrams (similar to MSCs, although less expressive than Z.120), collaboration diagrams (same information as MSCs, but with a two-dimensional view of the component architecture), and Statechart diagrams. The last type, *activity diagrams*, stands out as an interesting way of capturing scenarios. Activity diagrams capture the dynamic behavior of a system in terms of operations. They focus on end-to-end flows driven by internal processing. Activity diagrams share many characteristics with UCMs: focus on sequences of actions, guarded alternatives, and concurrency; complex activities can be refined; and simple mapping of behavior to components can be achieved through vertical *swimlanes*. However, activity diagrams do not capture dynamicity well, and the binding of actions to "components" is semantically weak in the current UML standard.

### 2.4 Summary of Evaluation

The thirteen scenario notations compared in this paper are summarized in Table 2. Due to some major differences, HMSCs are considered separately from basic MSCs in this table.

| *Scenario notation* | *Comp.-cent./ End-to-end* | *Hiding* | *Represen-tation* | *Ordering* | *Multiplicity* | *Abstraction* | *Identity* | *Dynamicity* |
|---|---|---|---|---|---|---|---|---|
| *MSC* | C-C | Yes | Sequence diagrams | Time | Single | Concrete | Many | Static |
| *HMSC* | C-C | Yes | Sequence diagrams | Causal | Multiple | Concrete | Many | Static |
| *Use Case* | C-C | No | Text | Time | Multiple | Abstract | Many | Static |
| *CREWS'* | C-C | No | Structured text | Time | Multiple | Abstract | Many | Static |
| *Scen. Tree* | C-C | No | Tree & grammar | Time | Multiple | Abstract | One | Static |
| *UCT* | C-C | No | Text & grammar | Time | Multiple | Concrete | One | Static |
| *Chisel* | C-C | No | Tree | Causal | Multiple | Abstract | Many | Static |
| *Statechart* | C-C | No | State machine | Causal | Multiple | Abstract | Many | Static |
| *LSC* | C-C | Yes | Sequence diagrams | Causal | Multiple | Concrete | Many | Static |
| *Somé's* | C-C | Yes | Structured text & Se-quence diagrams | Time | Multiple | Abstract | Many | Static |
| *RATS* | either | Yes | Structured text | Causal | Multiple | Abstract | Many | Static |
| *UCM* | E2E | Yes | Paths on components | Causal | Multiple | Abstract | Many | Dynamic |
| *UML Act. Diagrams* | E2E | Yes | Paths on swimlanes | Causal | Multiple | Abstract | Many | Static |

**Table 2** Evaluation of the Selected Scenario Notations

MSCs are most useful for single scenarios, especially when describing lengthy black-box interactions between actors and a given system (something that UML activity diagrams and UCMs do not do well). However, MSCs are not appealing for structuring related scenarios. HMSCs and LSCs are more powerful and expressive, but they still require an early commitment to components. Use cases and UCTs are generally not used to describe internal responsibilities and they do not support causal ordering. CREWS' scenarios improve on use cases by using structured text and guidelines, yet they have essentially the same limitations. Scenario trees and UCTs focus on only one actor at a time, which is often not desirable when describing telecommunications systems requirements. Chisel diagrams represent a good alternative to scenario trees, but they still focus on interactions between components. Somé's scenarios lack first-class causal ordering, which only appears when scenarios are transformed into component automata.

UCMs, RATS, and UML activity diagrams stand up as being the only surveyed scenario notations that are not component-centered but define end-to-end behavior. This is useful for early descriptions of requirements and helps to avoid overspecification. Also, UCMs stress causality relationships that can span many components. UML activity diagrams can, to some extent, present a similar view with swimlanes, but swimlanes are semantically weak and they cannot represent the architecture in two dimensions (swimlanes show components as columns). RATS scenarios can capture non-functional information, unlike most other notations. They have many of UCMs' characteristics, but UCMs have only one type of scenarios (not three as in RATS), and they are graphical, a property that makes them appealing to a variety of stakeholders. UCMs can also capture dynamicity through dynamic stubs (with multiple sub-maps selected at run-time) and dynamic responsibilities (which can move sub-maps around and store them in pools of sub-maps). This useful feature, fairly unique to UCMs, enables the description of emerging telecommunication services based on agents and dynamic selection of negotiation mechanisms.

The next section presents various construction approaches that target the construction of detailed behavior models from some of these scenario notations. These models are usually in the form of automata, (UML) statecharts, or formal specification languages such as ITU-T's SDL (Specification and Design Language) [43] and ISO's LOTOS (Language Of Temporal Ordering Specification) [39]. Such models are usually executable and suitable for validation, verification, and sometimes code generation.

# 3 Construction Approaches

## 3.1 Why Construction Approaches?

In the scenario-driven development of telecommunication systems and services, it is important to leverage the investment in scenarios in order to generate systems rapidly, at low cost, and with a high quality. To support the progression from scenarios capturing requirements and high-level functionalities to detailed designs and implementations based on communicating entities, we can learn much by examining different construction approaches used in the protocol engineering discipline, where the construction of a model based on another model is a concept supported by many techniques. In [63][68][69], Saleh and Probert present two categories of construction approaches for communication protocols that are also applicable to other areas of the telecommunication domain:

- **Analytic approach**: this is a build-and-test approach where the designer iteratively produces versions of the model by defining messages and their effect on the entities. Due to the manual nature of this construction approach, which often results in an incomplete and erroneous model, an extra step is required for the analysis, verification (testing), and correction of errors.
- **Synthetic approach**: a partially specified model is constructed or completed such that the interactions between its entities proceed without manifesting any error and (ideally) provide the set of specified services. For properties preserved by such approaches, no verification is needed as the correctness is insured by construction.

Synthetic approaches may or may not be fully automated. Sometimes, they require the *interactive* participation of the designer as some decisions need to be taken along the way. In both cases, synthetic approaches require the source model to be described formally (usually with some automata or with formal description techniques), whereas analytic approaches may start with semi-formal or informal models. In general, analytic and (interactive or automated) synthetic approaches have many other benefits and drawbacks, many of which are summarized in Table 3:

| | *Benefits* | *Drawbacks* |
|---|---|---|
| Analytic | • No formal source model required.<br>• Both the source and target models can exploit the richness and expressiveness of their respective modeling language to their full extent.<br>• The constructed model can more easily take into consideration design or implementation constraints (e.g. to reflect the high-level design), and be optimized accordingly.<br>• Non-functional requirements (e.g. performance, robustness) can more easily be taken into consideration. | • Transformation mostly manual.<br>• Errors may result from the construction.<br>• Verification is required.<br>• Many iterations may be required to fix the errors detected during verification.<br>• Time-consuming. |
| Synthetic, Interactive | • Improper synthetic constructions can be avoided by interacting with the designer.<br>• Correctness "ensured" by construction (under certain assumptions). Many faults are therefore avoided.<br>• Verification theoretically not required.<br>• Only one iteration required.<br>• Quick construction. | • Not fully automated.<br>• Requires formal and detailed source models.<br>• May require a partially constructed model to be available.<br>• Both source and target modeling languages are usually restricted in style and content.<br>• Difficult to take into consideration design/implementation constraints, optimizations, and non-functional requirements.<br>• Resulting model usually hard to understand, maintain and extend. |
| Synthetic, Automated | • Fully automated.<br>• Correctness "ensured" by construction (under certain assumptions). Many faults are therefore avoided.<br>• Verification theoretically not required.<br>• Only one iteration required.<br>• Very quick construction. | • Requires formal source models.<br>• Both source and target modeling languages are usually restricted in style and content.<br>• May result in improper synthetic constructions in ambiguous cases (the algorithm makes the decision, not the designer).<br>• Requires more details in the source model than non-automated approaches.<br>• Resulting model often hard to understand, maintain and extend. |

**Table 3** Benefits and Drawbacks of Construction Approaches

We have no intention of surveying the myriad of approaches for the synthesis of protocols or converters. However, we can build on the benefits and drawbacks presented here to evaluate construction techniques based on scenarios that are applicable to telecommunications systems in general.

### 3.2 Evaluation Criteria

The construction of models that integrate scenarios represents a problem similar to those faced by the protocol engineering community. A collection of scenarios often needs to be checked for completeness, consistency, and absence of undesirable interactions. To do so, most verification techniques require that a model that integrates these scenarios be available. Also, it is often desirable to map the scenarios onto a component architecture at design time in order to enable the generation of component behavior in distributed applications (e.g. telecommunication systems). These two construction levels are described below:

*C1)* Integration of a collection of requirements scenarios in an abstract model used for the analysis of requirements. No components are required here.

*C2)* Integration of a collection of scenarios in a component-based model used not only for the analysis of requirements, but also as a high-level design which considers some implementation issues.

Different approaches targeting these two levels are already available, nineteen of which are reviewed next. Additional evaluation criteria include:

- Type of construction approach: analytic, synthetic non-automated, or synthetic automated.
- Source scenario notation, such as the ones overviewed in Section 2.3.
- Whether the scenario model requires explicit components and messages.
- Target construction model (SDL, UML Statecharts, automata, LOTOS, etc.).

### 3.3 Selected Approaches

This section focuses on nineteen construction approaches particularly relevant to the telecommunication domain, and it provides a concise comparison in terms of the criteria seen in Section 3.2.

***Non-Automated Analytic Approaches***

- The *Usage Oriented Requirements Engineering* (UORE) approach proposed by **Regnell *et al.*** [64][65] builds on the Objectory methodology [47] and adds a construction phase (unfortunately called *synthesis* in that approach) where use cases are integrated manually into a *Synthesized Usage Model* (SUM). This "synthesis", which addresses level *C1*, is composed of three activities: formalization of use cases (using an extended MSC notation), integration of use cases (which produces usage views, one for each actor/component), and verification (through inspection and testing). The resulting SUM is a set of automata whose purpose is to serve as a reference model for design and V&V. No automated support is provided yet.
- In RATS, **Eberlein** provides informal guidelines [27]. Non-functional requirements have to be refined into either functional requirements or implementation constraints. The functional requirements have to be expressed in textual use cases. The user then has to define states in the system behavior. Adding pre-, flow- and post-conditions results in structured use cases. The most formal use-case notation uses atomic actions, which still contain textual descriptions. These formalized use cases are then mapped to SDL flowchart constructs in order to address level *C2*. The approach does not go deeply into the construction of the SDL model as RATS focuses more on the acquisition and the specification of requirements (including non-functional ones).
- **Bordeleau** addresses *C2* with the *Real-Time TRaceable Object-Oriented Process* (RT-TROOP) [17], which combines the use of scenario textual descriptions (use cases), UCMs, MSCs, and ROOM (now UML-RT) [73]. Included is an approach where UCM scenarios are first transformed into HMSCs, and then into hierarchical communicating finite state machines (ROOMCHARTS) [18]. No construction algorithm is proposed, but the use of transformation patterns is suggested instead. Several such patterns are provided for the UCM-HMSC mapping [19], and for the construction of ROOMCHARTS from HMSCs. HMSCs are used to fill the gap between UCMs, which abstract from message exchanges, and the state machines, which describe the behavior of the actors/components involved. Traceability relationships are also defined in this process. RT-TROOP focuses more on design than on requirements validation because verification of the ROOM model is limited. *ObjecTime*, ROOM's tool, only supports animation and a limited form of testing based on MSCs, but at the same time it supports automatic code generation.
- **Krüger *et al.*** present a related technique for the transformation of a set of MSCs to a Statechart model [53], hence addressing *C2*. The construction takes into consideration the type of semantics associated with MSCs, e.g. whether there are fewer, more, or the same number of components in the system than what is found in the MSCs, or whether additional messages (from another scenario) are allowed or forbidden between two messages in a component, etc. This technique is however very immature at this point and it is not supported by algorithms or tools.
- **Amyot** addresses *C2* with the *Specification and Validation Approach with LOTOS and UCMs* (SPECVALUE) [7][8], where UCMs describe functional scenarios bound to architectural components and LOTOS formalizes the integration of scenarios and the distribution of behavior over communicating entities. LOTOS is a formal language that has constructs similar to those found in the UCM notation,

and it complements UCMs weaknesses in the area of executability and verification. Guidelines are provided for the manual construction of LOTOS models from UCMs, but no automation is provided mainly because of the semi-formal nature of UCMs and of the design decisions required to refine causal paths in terms of message exchanges. SPEC-VALUE also includes a pattern language for the manual extraction of test cases used to validate the LOTOS model against the UCMs, and hence against the functional requirements. The LOTOS testing theory and tools are used to perform this validation, to provide coverage measures, and to detect unexpected and undesirable interactions between the scenarios.

- According to **Lamsweerde and Willemet**, a drawback of scenarios is that system properties are often left implicit. If these properties were explicit (e.g. in declarative terms), then consistency/completeness analysis would be much easier to carry out. Lamsweerde and Willemet address *C1* by exploring the process of inferring (by induction) formal specifications of such properties (goals) from scenario descriptions [54]. Their scenarios are sequential interaction diagrams whereas their goals are linear temporal logic (LTL) properties expressed in the KAOS language. The scenarios can either be positive (must be covered) or negative (must be excluded). Their technique represents a novel and promising contribution, but it remains analytic: it requires validation to be performed because inductive inference is not sound. This approach is not yet supported by tools.

***Non-Automated Synthesis Approaches***

- **Desharnais *et al*.** propose a synthesis approach for the integration of sequential scenarios represented in state-based relational algebra [25]. The initial scenarios involve the system and a single actor (concurrency is not considered), and the result is one large scenario represented again in relational algebra (thus *C1* is addressed). Although the authors claim that data and complex conditions being incorporated in the formalism represent an advantage over other approaches, their technique appears somewhat limited in terms of usability and scalability for realistic telecommunication systems.
- **Somé** proposes a composition algorithm that transforms his scenarios into Alur's timed automata [7], one for each component (hence addressing *C2*) [74][75]. This synthesis algorithm is implemented in a prototype tool, where consistency and completeness issues in the scenarios are resolved through the interactive assistance of the requirements engineer. The synthesis is based on the common preconditions rather than on the sequences of actions. Super-states are used when the preconditions of one scenario are included in that of a second scenario. The algorithm preserves the temporal constraints associated to the scenarios, which is seldom the case of other (semi-)automated synthesis techniques.
- **Harel and Kugler** propose an algorithm for the synthesis of Statecharts from a subset of the Life Sequence Chart (LSC) notation, without data or conditions [30][32]. This algorithm decides the satisfyability and consistency of a set of LSCs, something that is harder to do than for MSCs due to the possibility of expressing forbidden scenarios. The algorithm then produces a global system automaton. In order to address *C2*, this global automaton can be distributed (as Statecharts) over the set of components involved in the LSCs. These components share all their information with each other, which simplifies the synthesis algorithm. This work is promising but it is not yet supported by tools.
- **Alur *et al.*** have an algorithm that transforms a set of stateless basic MSCs into communicating state machines of various types (*C2*) [4]. This technique supports the detection of implied scenarios resulting from the composition of multiple MSCs. Alur's algorithm uses a language-theoretic framework with closure conditions. Its emphasis is on safety and on efficiency (it executes in polynomial time), and it can generate counter-examples for non-realizable sets of MSCs. The detection is based on previous work done in collaboration with Holzmann and Peled [3], who extended this work in another direction to support HMSCs during requirements analysis with the tool *uBET* [35].

***Automated Synthesis Approaches***

- With their SCED methodology [51], **Koskimies *et al*.** propose a synthesis algorithm that integrates *scenario diagrams*, an extension of the basic MSC'92 notation with iterations, conditions, and subscenarios (thus more in line with the MSC 2000 standard). The algorithm outputs OMT state dia-

grams, which are based on Harel's Statecharts. The synthesis is supported by the SCED tool [52], which also contains visual editors for scenario diagrams and state diagrams. The state machine generated by the tool is minimal with respect to the number of states necessary to support the scenarios. The authors claim that their approach is not tied to the OMT methodology, and hence can be reused in other contexts to address *C2*.

- **Schönberger *et al*.** have developed another algorithm based on a similar idea [71], only this time they start with another type of scenario notation: UML collaboration diagrams. Their synthesis procedure addresses *C2* by generating UML Statecharts, which make extensive use of concurrency constructs to satisfy the inherent concurrency found in collaboration diagrams (but absent from Koskimies' scenario diagrams). Although their algorithm does not output a minimized state machine, the authors provide several state diagram compression techniques. This procedure has a polynomial complexity and is not incremental, whereas Koskimies' approach is incremental but with an exponential complexity. A prototype tool implements this algorithm and can be used to generate graphical user interfaces automatically, provided that the initial collaboration diagrams are enriched with necessary user interface information (e.g. selection of buttons, display of text fields, etc.) [28].
- **Whittle and Shumann** propose an algorithm for the generation of UML Statecharts from a collection of UML sequence diagrams (*C2*) [79]. It allows for conflicts to be detected and resolved through UML's *Object Constraint Language* (OCL) and global state variables. These Statecharts can be non-deterministic. The target Statechart model is intended to be highly structured (hierarchical) and readable in order to be modified and refined by designers. This algorithm shares similarities with the work of Schönberger [71] and Somé [74] as the hierarchical nature of the states is inferred. However, the synthesis is also influenced by structure elements found in other types of UML diagrams such as class diagrams. The approach is supported by a prototype tool written in Java.
- **Leue *et al*.** have developed two algorithms for the automated synthesis of Real-Time Object-Oriented Modeling (ROOM) models from standard HMSC scenarios [55]. Essentially, ROOMCHARTS are generated for each actor in the HMSCs, hence addressing *C2*. One major assumption is that the basic MSCs referenced by the HMSC are mutually exclusive, i.e. unlike SCED, only one scenario is active at any time. This results in simpler synthesis algorithms, at the cost of a major limitation for describing realistic telecommunication systems. The first algorithm, called *maximum traceability*, preserves the HMSC structure in the synthesized model. The second one, called *maximum progress*, generates smaller state machines but sacrifices traceability with respect to HMSCs. The properties preserved by these algorithms are still under investigation. Both algorithms are implemented in the MESA toolset [13], and their authors claim that their work can be adapted to support SDL and UML.
- **Mansurov and Zhukov** address *C2* and target the automated generation of SDL models from HMSCs [59]. The scenarios are first sliced by actor, and then communicating finite state machines are generated for each actor. These FSMs are made deterministic and minimal, and then transformed into SDL processes. The resulting SDL system usually allows more traces than those defined by the HMSCs. Very little is said about the synthesis algorithm itself, and the levels of detail and consistency required by the MSCs is relatively high. This technique is implemented in MOST, the *Moscow Synthesizer Tool*.
- **Li and Horgan** target the architectural analysis of telecommunications systems with an algorithm for the semi-automated synthesis of SDL models from architectures described using component, links, and *archflows* [56]. Archflows are sequential workflows where the steps are observable events, internal events, or sending/reception of messages performed by components (hence addressing *C2*). The resulting SDL model is complete and assumed to be valid when it contains all the archflow traces. Workflows are assumed not to conflict with each other, hence they should be consistent and have no undesirable interaction, which is of limited use for early validation. Non-determinism is allowed, and the model can be supplemented with performance information for performance prediction evaluations. The method is supported by a toolset, the *Workflow-to-SDL-Direct-Simulation*.
- **Khendek and Vincent** propose an approach for the incremental construction of an SDL model given an existing SDL model, whose properties need to be preserved (an extension relation is provided), and a set of new MSC scenarios [49]. The synthesis algorithm considers only input/output signals, not

the actions in the transitions. The semi-automated construction is done in three steps: add new components if necessary (manually), synthesize the new architecture behavior from MSCs using the MSC2SDL tool [1], and then merge the behavior descriptions of the old SDL with the increment SDL, on a per process basis. If non-determinism that violates the extension relation is added along the way, then the tool reports the problem (error detection only). If an MSC description of the old SDL specification is available, then the approach can be simplified to adding new MSCs to the old MSCs and regenerate the new specification using the MSC2SDL tool. However, the extension relation may also be violated by this approach.

- **Turner** presents an approach called CRESS (*Chisel Representation Employing Systematic Specification*), which defines tightly defined rules for the syntax and static semantics of an enhanced version of Chisel diagrams [76]. This improved notation has formal denotations in both LOTOS and SDL, hence enabling the synthesis of formal models in order to support the rapid creation, specification, analysis and development of features. Although CRESS often represents scenarios as trees (more precisely as directed acyclic graphs), the tree nodes represent interactions between components. Hence, this approach is roughly comparable to the ones starting from HMSCs (although CRESS' interactions are synchronous and directionless) and it also addresses *C2*. CRESS is supported by a set of tools for parsing, checking and translating diagrams. However, the synthesis algorithm remains undocumented and hence little is known about the design decisions taken by the translation tools.
- **Dulz *et al***. present an approach where performance prediction models (in SDL) are also automatically synthesized from MSC scenarios, but this time supplemented with performance annotations [26]. Their goal is to obtain performance estimates early in the design process (various other techniques for the construction of performance models from UML and SDL are reported in [80]). The synthetic SDL model (addressing *C1*) is intended to be a throwaway prototype, but it is nonetheless used to generate the code for the target system whose performance is evaluated. The approach is supported by a prototype tool (LISA), however the algorithm remains obscure; it is not even clear whether two MSCs that start with a similar transition should be composed as alternatives, as sequences, or in parallel.

### 3.4 Summary of Evaluation

Several aspects of the construction approaches reviewed in Section 3.3 are summarized in Table 4. These aspects correspond to the evaluation criteria specified in Section 3.2.

Most of the techniques surveyed here require the use of scenario notations based on messages exchanged between communicating entities (see column *Component-Based* in Table 4). MSC-like notations such as basic MSCs, extended MSCs, HMSCs, LSCs, and UML interaction diagrams are especially common as source scenario models for construction approaches. Techniques based on HMSCs can further benefit from recent theoretical results on necessary conditions for the synthesis of communicating automata from HMSCs [33]. For target construction models, communicating finite state machines, whether they are hierarchical (ROOMCHARTS, (UML) Statecharts, or OMT state diagrams) or not (SDL or plain CFSMs) are very common. It is difficult to determine the best construction approach for component-based scenarios as they use varying source and target models, they are still under heavy development, and they are not supported by commercial tools. Synthesis approaches also have different sets of constraints and design decisions embedded in their algorithms.

Only three of the techniques surveyed (RATS, RT-TROOP, and SPEC-VALUE) do not start from scenarios expressed in terms of components and messages, and they are only used in analytic construction approaches. In SPEC-VALUE and RT-TROOP, UCMs abstract from the communication aspect between the components. UCMs are hence less coupled to a specific architecture and closer to the functional requirements. However, the information related to exchanges of messages (e.g. protocols and negotiation mechanisms) needs to be provided while constructing a target component-based model.

| *Construction Approach* | *Level C1/C2* | *Type of Approach* | *Scenario Models* | *Comp.-Based* | *Construction Models* |
|---|---|---|---|---|---|
| Regnell et al. (UORE) | *C1* | *Analytic* | *Extended MSC* | *Y* | *Automata* |
| Eberlein (RATS) | *C2* | *Analytic* | *Structured text* | *N* | *SDL* |
| Bordeleau (RT-ROOP) | *C2* | *Analytic* | *UCMs, HMSCs* | *N* | *ROOMCHARTS* |
| Krüger et al. | *C2* | *Analytic* | *MSCs* | *Y* | *Statecharts* |
| Amyot (SPEC-VALUE) | *C2* | *Analytic* | *UCMs* | *N* | *LOTOS* |
| *Lamsweerde* & Willemet | *C1* | *Analytic* | *Sequential and synchronous MSCs* | *Y* | *LTL properties in KAOS* |
| *Desharnais* et al. | *C1* | *Synthetic, non-automated* | *State-based relational algebra* | *Y* | *State-based relational algebra* |
| Somé | *C2* | *Synthetic, non-automated* | *Structured text and extended MSCs* | *Y* | *Timed automata* |
| Harel and Kugler | *C2* | *Synthetic, non-automated* | *LSCs* | *Y* | *Statecharts* |
| Alur et al. | *C2* | *Synthetic, non-automated* | *Basic MSCs* | *Y* | *CFSMs* |
| Koskimies et al. (SCED) | *C2* | *Synthetic, automated* | *Extended MSCs* | *Y* | *OMT state diagrams* |
| Schönberger et al. | *C2* | *Synthetic, automated* | *UML collaboration diagrams* | *Y* | *UML Statecharts* |
| Whittle and Schumann | *C2* | *Synthetic, automated* | *UML sequence diagrams* | *Y* | *UML Statecharts* |
| *Leue et al.* | *C2* | *Synthetic, automated* | *HMSCs* | *Y* | *ROOMCHARTS* |
| *Mansurov* and Zhukov | *C2* | *Synthetic, automated* | *HMSCs* | *Y* | *SDL* |
| Li and Horgan | *C2* | *Synthetic, automated* | *Archflows* | *Y* | *SDL* |
| *Khendek* and Vincent | *C2* | *Synthetic, automated* | *MSCs, SDL* | *Y* | *SDL* |
| Turner (CRESS) | *C2* | *Synthetic, automated* | *Extended Chisel diagrams* | *Y* | *SDL or LOTOS* |
| *Dulz et al.* | *C1* | *Synthetic, automated* | *Extended MSCs* | *Y* | *SDL* |

**Table 4** Evaluation of the Selected Construction Approaches

An interesting characteristic of UCMs and LOTOS languages is that they can both address *C1* and *C2*. UCM scenario paths, like UML activity diagrams but unlike component-based notations, do not require the presence of any entity to be meaningful. Similarly, LOTOS is abstract enough to specify scenario behavior without any reference to components. CFSM-like modeling languages (Statecharts, SDL, etc.) usually require the explicit definition of such components. SPEC-VALUE is therefore fairly unique in that it enables the construction of executable and validatable models in the presence or in the absence of components. This is particularly useful in the early stages of the design process where the architecture is still undefined, or when it is desirable to remain independent from any architecture. Later in the design process, an architecture may become available and both models (UCM scenarios and LOTOS) can be adapted accordingly. Unfortunately, due to the nature of UCMs, such an approach is difficult to automate to a large extent.

## 4 Hopes and Challenges

Scenario-based modeling, analysis, and transformations are currently the focus of much attention and research activities. Conferences dedicated exclusively to these topics have started to appear, particularly in the UML, requirements engineering, and formal methods communities. Industrial interest transpires through tool builders, user groups, and various standardization committees at the ITU-T (International Telecommunications Union) and at the OMG (Object Management Group), to name a few. In order for a scenario notation to become widely accepted in industry, good tool support is required together with interfaces to other existing notations and tools.

Current activities at the OMG include the integration of Message Sequence Charts (Z.120) in UML, in order to replace the current sequence diagrams. In fact, the OMG and ITU-T Study Group 10 are working towards the integration of their scenario-based approaches and notations. The recent SDL with UML standard (Z.109 [44]) is one step in that direction, and many others are to come. Hopefully, this will lead to better and more unified approaches for the synthesis of component-based models (SDL, Statecharts, etc.) from message-based scenarios. Such approaches could improve the penetration and acceptance of formal methods and of new requirements, design, and analysis techniques.

ITU-T Study Group 10 has also started work towards the standardization of a *User Requirements Notation* (URN) [46], which targets the representation of requirements for future telecommunication systems and services. At this early stage of its development, URN contains one notation for the representation of non-functional requirements (NFRs) and a complementary scenario notation for functional requirements. NFRs are seldom captured explicitly in design processes (even in UML) [22], and URN is a first standardization effort in that direction. The scenario notation is based on Use Case Maps and hence abstracts from message exchanges. URN fits nicely in the very first stage of existing ITU-T methodologies (such as I.130 [41] and Q.65 [42]) and smoothes the transition towards later stages of development processes. On top of traceability links between NFRs and UCMs, the proposed standard intends to provide additional insights on the derivation of MSCs [6][19] and the generation of formal models [7][8][18][70] and performance models [72] from URN models. Links to the UML standard will also be investigated in order to unify common concepts and perhaps notations [9]. Tool support is also being investigated [60].

It is our hope that future development processes for telecommunication systems will make good use of standardized scenario notations and construction approaches. Current and emerging work and communities focusing on scenarios and their transformation represent a major step in that direction. One of the main challenges now consists in establishing common grounds and objectives so that potential standards such as URN may eventually become reality.

## 5 Conclusions

The development of current and emerging telecommunication systems can benefit today from scenario notations, and more so at the requirements stage. Due to prolific and enthusiastic researchers worldwide, various scenario approaches are now available.

Section 2 focused on twelve state-of-the-art notations and provided an evaluation against eight criteria relevant to the telecommunication domain. Most of the reviewed notations are centered around messages exchanged between system components whereas a few focus on end-to-end behavior, independently of component architectures. The former are most useful for describing lengthy scenarios involving external actors and the system as a unique component, and for detailed design involving multiple system components. The latter are more appropriate for capturing functional requirements without committing to specific architectures and protocols.

The transition from scenario models to more detailed design and implementation-oriented models is discussed in Section 3. Nineteen analytic and synthetic construction approaches are reviewed and briefly compared. Most of them target the generation of component-based design models from scenario models that are also component-based. A handful of construction approaches attempt to bridge the gap between requirements and design by abstracting from message exchanges, but they are analytic and lack formal algorithms for conversion.

Even if many challenges remain, it is our hope that research will produce better scenario notations and construction approaches, which will be widely adopted in industrial practices. The high interest demonstrated towards scenarios for telecommunication systems and towards emerging standards like the User Requirements Notation is certainly a good indication that industry and academia are heading in the right direction.


### Acknowledgement

We are most thankful to Prof. Luigi Logrippo and to Gunter Mussbacher for useful comments on an earlier version of this evaluation. This work was supported in part by NSERC, FCAR, CITO, Mitel Corporation, and Nortel Networks.


## 6 References


[1] Abdalla, M.M., Khendek, F. and Butler, G. (1999) "New Results on Deriving SDL Specifications from MSCs". In: SDL'99, Proceedings of the Ninth SDL Forum, Montréal, Canada. Elsevier.

[2] Aho, A., Gallagher, S., Griffeth, N., Scheel, C., and Swayne, D. (1998) "Sculptor with Chisel: Requirements Engineering for Communications Services". In: K. Kimbler and L. G. Bouma (Eds), *Fifth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, Lund, Sweden, September 1998. IOS Press, 45-63. http://www-db.research.bell-labs.com/user/nancyg/sculptor.ps

[3] Alur, R. Holzmann, G. and Peled, D. (1996) "An Analyzer for Message Sequence Charts". In: *Software Concepts and Tools*, 17(2):70-77. http://cm.bell-labs.com/cm/cs/what/ubet/papers/aAfMSCs.ps.gz

[4] Alur, R., Etessami, K., and Yannakakis, M. (2000) "Inference of Message Sequence Charts". In: *22th International Conference on Software Engineering* (ICSE 2000), Limerick, Ireland, ACM Press, 304-313.

[5] Amyot, D., Andrade, R., Logrippo, L., Sincennes, J., and Yi, Z. (1999) "Formal Methods for Mobility Standards". In: *IEEE 1999 Emerging Technology Symposium on Wireless Communications & Systems*, Richardson, Texas, USA, April 1999. http://www.UseCaseMaps.org/pub/ets99.pdf

[6] Amyot, D. and Andrade, R. (1999) "Description of Wireless Intelligent Network Services with Use Case Maps". In: *SBRC'99, 17° Simpósio Brasileiro de Redes de Computadores*, Salvador, Brazil, May 1999, 418-433. http://www.UseCaseMaps.org/pub/sbrc99.pdf

[7] Amyot, D., Buhr, R.J.A., Gray, T., and Logrippo, L. (1999) "Use Case Maps for the Capture and Validation of Distributed Systems Requirements". In: *RE'99, Fourth IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, 44-53. http://www.UseCaseMaps.org/pub/re99.pdf

[8] Amyot, D. and Logrippo, L. (2000) "Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System". In: *Computer Communication*, 23(12), 1135-1157. http://www.UseCaseMaps.org/pub/cc99.pdf

[9] Amyot, D. and Mussbacher, G. (2000) "On the Extension of UML with Use Case Maps Concepts". In: *<<UML>>2000, 3rd International Conference on the Unified Modeling Language*, York, UK, October 2000. LNCS 1939, 16-31. http://www.UseCaseMaps.org/pub/uml2000.pdf

[10] Amyot, D. (2000) "Use Case Maps as a Feature Description Language". In: S. Gilmore and M. Ryan (Eds), *Language Constructs for Designing Features*. Springer-Verlag, 27-44. http://www.UseCaseMaps.org/pub/fireworks2000.pdf

[11] Andersson, M. and Bergstrand, J. (1995) *Formalizing Use Cases with Message Sequence Charts*. Master thesis, Department of Communication Systems, Lund Institute of Technology, Sweden, May 1995. http://www.efd.lth.se/~d87man/EXJOBB/Title_Abstract_Preface.html

[12] Ardis, M.A., Chaves, J.A., Jagadeesan, L. J., Mataga, P., Puchol, C., Staskauskas, M.G., and Olnhausen, J.V. (1996) "A Framework for Evaluating Specification Methods for Reactive Systems — Experience Report". In: *IEEE Transactions on Software Engineering*, 22 (6), 378-389.

[13] Ben-Abdallah, H. and Leue, S. (1997) *MESA*: *Support for scenario-based design of concurrent systems*. Technical Report 97-12, Department of Electrical & Computer Engineering, University of Waterloo, Canada, October.

[14] Ben Achour, C., Rolland, C., Maiden, N.A.M., and Souveyet, C. (1999) "Guiding Use Case Authoring: Results of an Empirical Study". In: *RE'99, Fourth IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, 36-43.

[15] Benner, K.M., Feather, M.S., Johnson, W.L., and Zorman, L.A. (1993) "Utilizing Scenarios in the Software Development Process". In: *Information System Development Process*, Elsevier Science, B.V. North-Holland, 117-134.

[16] Boni Bangari, A. (1997) *A Use Case Driven Validation Framework and Case Study*. M.Sc. thesis, SITE, University of Ottawa, Ottawa, Canada.

[17] Bordeleau, F. and Buhr, R.J.A. (1997) "The UCM-ROOM Design Method: from Use Case Maps to Communicating State Machines". In: *Conference on the Engineering of Computer-Based Systems*, Monterey, USA, March 1997. http://www.UseCaseMaps.org/pub/UCM-ROOM.pdf

[18] Bordeleau, F. (1999) *A Systematic and Traceable Progression from Scenario Models to Communicating Hierarchical Finite State Machines*. Ph.D. thesis, School of Computer Science, Carleton University, Ottawa, Canada. http://www.UseCaseMaps.org/pub/fb_phdthesis.pdf

[19] Bordeleau, F. and Cameron, D. (2000) "On the Relationship between Use Case Maps and Message Sequence Charts". In: *2nd Workshop of the SDL Forum Society on SDL and MSC (SAM2000)*, Grenoble, France, June 2000. http://www.UseCaseMaps.org/pub/sam2000.pdf

[20] Buhr, R.J.A. (1998) "Use Case Maps as Architectural Entities for Complex Systems". In: *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*. Vol. 24, No. 12, December 1998, 1131-1155. http://www.UseCaseMaps.org/pub/tse98final.pdf

[21] Chandrasekaran, P. (1997) "How Use Case Modeling Policies Have Affected the Success of Various Projects (Or How to Improve Use Case Modeling)". In: *Addendum to the 1997 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA'97)*, 6-9.

[22] Chung, L., Nixon, B.A., Yu, E. and Mylopoulos, J. (2000) *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers.

[23] Cockburn, A. (1997) “Structuring Use cases with goals”. In: *Journal of Object-Oriented Programming (JOOP/ROAD)*, 10(5), September 1997, 56-62. http://members.aol.com/acockburn/papers/usecases.htm

[24] Damm, W. and Harel, D. (1999) "LCSs: Breathing Life into Message Sequence Charts". In: *3rd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'99)*, Kluwer Academic Publishers, 293-312.

[25] Desharnais, J., Frappier, M., Khédri, R., and Mili, A. (1997). "Integration of Sequential Scenarios". In: *ESEC'97, Sixth European Engineering Conference*, LNCS 1301, Springer-Verlag, 310-326.

[26] Dulz, W., Gruhl, S., Lambert, L., and Söllner, M. (1999) "Early performance prediction of SDL/MSC specified systems by automated synthetic code generation". In: *SDL'99, Proceedings of the Ninth SDL Forum*, Montréal, Canada. Elsevier.

[27] Eberlein, A. (1997) *Requirements Acquisition and Specification for Telecommunication Services*. PhD thesis, University of Wales, Swansea, UK. http://www.enel.ucalgary.ca/People/eberlein/publications/thesis.zip

[28] Elkoutbi, M.., Khriss, I., and Keller, R.K. (1999) "Generating User Interface Prototypes from Scenarios". In: *RE'99, Fourth IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, 150-158. ftp://ftp.iro.umontreal.ca/pub/gelo/Publications/Papers/isre99.pdf

[29] Glinz, M. (1995) "An Integrated Formal Model of Scenarios Based on Statecharts". In: *Proceedings of the 5th European Software Engineering Conference (ESEC 1995)*, Sitges, Spain.

[30] Harel, D. (2000) "From Play-In Scenarios To Code: An Achievable Dream". In: *Fundamental Approaches to Software Engineering (FASE2000)*, LNCS 1783, Springer-Verlag, 22-34. http://www.wisdom.weizmann.ac.il:81/Dienst/UI/2.0/Describe/ncstrl.weizmann_il/MCS00-06

[31] Harel, D. and Gery, E. (1996) "Executable Object Modeling with Statecharts". In: *Proceedings of the 18th International Conference on Software Engineering*, Berlin, IEEE Press, March 1996, 246-257.

[32] Harel, D. and Kugler, H. (2000) "Synthesizing State-Based Object Systems from LSC Specifications". In: *Fifth International Conference on Implementation and Application of Automata (CIAA 2000)*, LNCS, Springer-Verlag.

[33] Hélouët, L. and Jard, C. "Conditions for synthesis of communicating automata from HMSCs". In: *5th International Workshop on Formal Methods for Industrial Critical Systems*, Berlin, April 2000. http://www.fokus.gmd.de/research/cc/tip/fmics/abstracts/helouet.html

[34] Hodges, J. and Visser, J. (1999) "Accelerating Wireless Intelligent Network Standards Through Formal Techniques". In: *IEEE 1999 Vehicular Technology Conference (VTC'99)*, Houston (TX), USA. http://www.UseCaseMaps.org/pub/vtc99.pdf

[35] Holzmann, G.J., Peled, D., and Redberg, M. (1997) "Design tools for requirements engineering". In: *Bell Labs Technical Journal*, 2(1):86-95. http://www.lucent.com/minds/techjournal/winter_97/pdf/paper07.pdf http://cm.bell-labs.com/cm/cs/what/ubet/

[36] Hsia, P., Samuel, J., Gao, J., Kung, D., Toyoshima, Y. and Chen, C. (1994) "Formal Approach to Scenario Analysis". In: *IEEE Software*, 1994, 33-40.

[37] Hurlbut, R. (1997) *A Survey of Approaches for Describing and Formalizing Use Cases*. Technical Report 97-03, Department of Computer Science, Illinois Institute of Technology, USA. http://www.iit.edu/~rhurlbut/xpt-tr-97-03.html

[38] Hurlbut, R. R. (1998) *Managing Domain Architecture Evolution Through Adaptive Use Case and Business Rule Models*. Ph.D. thesis, Illinois Institute of Technology, Chigago, USA. http:// www.iit.edu/~rhurlbut/hurl98.pdf

[39] ISO (1989), Information Processing Systems, Open Systems Interconnection, *LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807, Geneva.

[40] ISO/EIC (1997) *OSI CTMF Part 3: The Tree and Tabular Combined Notation — Second Edition*, IS 9646-3: 1997, Geneva.

[41] ITU-T (1988) *Recommendation I.130, Method for the characterization of telecommunication services supported by an ISDN and network capabilities of ISDN*. CCITT, Geneva.

[42] ITU-T (2000) *Recommendation Q.65, The unified functional methodology for the characterization of services and network capabilities including alternative object-oriented techniques*. Geneva.

[43] ITU-T (2000) *Recommendation Z.100, Specification and Description Language (SDL).* Geneva.

[44] ITU-T (2000) *Recommendation Z.109, SDL combined with UML*. Geneva.

[45] ITU-T (2000) *Recommendation Z. 120: Message Sequence Chart (MSC).* Geneva.

[46] ITU-T, Study Group 10 (1999) *Proposal for a new question to define a notation for user requirements*. Canadian contribution, COM10-D56, November 1999.

[47] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1993) *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, ACM Press.

[48] Jarke, M. and Kurki-Suonio, R., editors. (1998) *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*. Vol. 24, No. 12, December 1998.

[49] Khendek, F. and Vincent, D. (2000) "Enriching SDL Specifications with MSCs". In: *2nd Workshop of the SDL Forum Society on SDL and MSC (SAM2000)*, Grenoble, France, June 2000.

[50] Kimbler, K. and Søbirk, D. (1994) "Use case driven analysis of feature interactions". In: L. G. Bouma and H. Velthuijsen (eds), *Feature Interactions in Telecommunications Systems*, Amsterdam, The Netherlands, May 1994. IOS Press, 167-177.

[51] Koskimies, K. and Mäkinen, E. (1994) "Automatic Synthesis of State Machines from Trace Diagrams". In: *Software Practice and Experience*, 24(7), July 1994, 643-658.

[52] Koskimies, K., Männistö, T., Systä, T., and Tuomi, J. (1996) *SCED: A tool for dynamic modelling of object systems*. University of Tampere, Department of Computer Science, Report A-1996-4, July. ftp://cs.uta.fi/pub/reports/A-1996-4.ps.Z

[53] Krüger, I., Grosu, R., Scholz, P. and Broy, M. (1999) "From MSCs to Statecharts". In: *Distributed and Parallel Embedded Systems*, Kluwer Academic Publishers. http://www4.informatik.tu-muenchen.de/papers/KGSB99.html

[54] van Lamsweerde, A. and Willemet, L. (1998) "Inferring Declarative Requirements Specifications from Operational Scenarios". In: *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*. Vol. 24, No. 12, Dec. 1998, 1089-1114.

[55] Leue, S., Mehrmann, L. and Rezai, M. (1998) "Synthesizing ROOM Models from Message Sequence Chart Specifications". Technical Report 98-06, ECE Dept., University of Waterloo, Canada, April 1998. Short paper version in: *13th IEEE Conference on Automated Software Engineering*, Honolulu, Hawaii, October 1998. http://sven.uwaterloo.ca:80/~sleue/publications.files/tr98-06.ps.gz

[56] Li, J.J. and Horgan, J.R. (2000) "Applying formal description techniques to software architectural design". In: *Computer Communications*, 23(12), 1169-1178.

[57] Lucent Technologies (1999) *uBET — Lucent Behavior Engineering Toolset*. http://cm.bell-labs.com/cm/cs/what/ubet/

[58] Maiden, N.A.M. (1998) "SAVRE: Scenarios for Acquiring and Validating Requirements". In: *Journal of Automated Software Engineering*, 5, 419-446.

[59] Mansurov, N. and Zhukov, D. (1999) "Automatic synthesis of SDL models in use case methodology". In: *SDL'99, Proceedings of the Ninth SDL Forum*, Montréal, Canada. Elsevier.

[60] Miga, A. (1998) *Application of Use Case Maps to System Design with Tool Support*. M.Eng. thesis, Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada. http://www.UseCaseMaps.org/pub/am_thesis.pdf

[61] OMG (1999) *Unified Modeling Language Specification, Version 1.3*. June 1999. http://www.omg.org

[62] Potts, C., Takahashi, K., and Antòn, A.I. (1994) "Inquiry-Based Requirements Analysis". In: *IEEE Software*, March 1994, 21-32.

[63] Probert, R.L. and Saleh, J. (1991) “Synthesis of communications protocols: survey and assessment”. In: *IEEE Transactions on Computers*, Vol. 40, No. 4, April 1991, 468-476.

[64] Regnell, B., Kimbler, K., and Wesslén, A. (1995) "Improving the Use Case Driven Approach to Requirements Engineering". In: *Proceedings of Second IEEE International Symposium on Requirements Engineering*, York, U.K., March 1995, 40-47. http://www.tts.lth.se/Personal/bjornr/Papers/tts-94-24.ps

[65] Regnell, B. (1999) *Requirements Engineering with Use Cases — a Basis for Software Development*. Ph.D. Thesis, Department of Communication Systems, Lund Institute of Technology, Sweden. http://www.tts.lth.se/Personal/bjornr/thesis/

[66] Rolland, C., Ben Achour, C., Cauvet, C., Ralyte, J., Sutcliffe, A.G., Maiden, N.A.M., Jarke, M., Haumer, P., Pohl, K., Dubois, E., and Heymas, P. (1998) "A proposal for a Scenario Classification Framework". *Requirements Engineering Journal*, 3(1), 23-47.

[67] Rolland, C., Souveyet, C. and Ben Achour, C. (1998) "Guiding Goal Modelling using Scenarios". In: *IEEE Transactions on Software Engineering, Special Issue on Scenario Management*. Vol. 24, No. 12, December 1998.

[68] Saleh, K. (1996) "Synthesis of communications protocols: an annotated bibliography". In: *ACM SIGCOMM Computer Communications Review*, Vol.26 , No.5, October, 40-59.

[69] Saleh, K. (1998) "Synthesis of protocol converters: an annotated bibliography". *Computer Standards & Interfaces*, 19, 105-117.

[70] Sales, I. and Probert, R. (2000) "From High-Level Behaviour to High-Level Design: Use Case Maps to Specification and Description Language". In: *SBRC'2000, 18° Simpósio Brasileiro de Redes de Computadores*, Belo Horizonte, Brazil, May 2000.

[71] Schönberger, S., Keller, R.K., and Khriss, I. (1999) "Algorithmic Support for Model Transformation in Object-Oriented Software Development". In: *Theory and Practice of Object Systems (TAPOS)*. John Wiley and Sons. ftp://ftp.iro.umontreal.ca/pub/gelo/Publications/Papers/tapos99.pdf

[72] Scratchley, W.C. (2000) *Evaluation and Diagnosis of Concurrency Architectures*. Ph.D. thesis, Carleton University, Ottawa, Canada. http://www.UseCaseMaps.org/pub/scratchley-thesis.pdf

[73] Selic, B., Gullekson, G., and Ward, P.T. (1994) *Real-Time Object-Oriented Modeling*, Wiley & Sons.

[74] Somé, S., Dssouli, R., and Vaucher J. (1996) "Toward an Automation of Requirements Engineering using Scenarios". In: *Journal of Computing and Information*, 2(1), 1110-1132.

[75] Somé, S. (1997) *Dérivation de Spécifications à partir de Scénarios d'interaction*. Ph.D. thesis, Département d'IRO, Université de Montréal, Canada.

[76] Turner, K.J. (2000) "Formalising the Chisel Feature Notation"'. In: *Sixth International Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, Glasgow, Scotland, UK, May 2000. IOS Press, Amsterdam, 241-256. ftp://ftp.cs.stir.ac.uk/pub/staff/kjt/research/pubs/form-chis.pdf

[77] *Use Case Maps Web Page* and *UCM User Group* (1999). http://www.UseCaseMaps.org

[78] Weidenhaupt, K., Pohl, K., Jarke, M., and Haumer, P. (1998) "Scenarios in System Development: Current Practice". In: *IEEE Software*, March/April 1998, 34-45.

[79] Whittle, J. and Shumann, J. (2000) "Generating Statechart Designs From Scenarios". In: *22th International Conference on Software Engineering* (ICSE 2000), Limerick, Ireland, ACM Press, 314-323.

[80] Woodside, C.M., Menascé, D. and Gomaa, H. eds. (2000) *Proceedings of the Second International Workshop on Software and Performance (WOSP'2000)*, Ottawa, Canada, September. http://www.sce.carleton.ca/wosp2000/

[81] Zave, P. and Jackson, M. (1997) "Four dark corners of requirements engineering". In: *ACM Transactions on Software Engineering and Methodology VI(1)*, January 1997, 1-30. http://www.research.att.com/~pamela/4dc.ps