



An Evaluation of Scenario Notations and Construction Approaches for Telecommunication Systems Development

DANIEL AMYOT

damyot@site.uottawa.ca

School of Information Technology and Engineering, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada

ARMIN EBERLEIN

eberlein@enel.ucalgary.ca

Department of Electrical & Computer Engineering, University of Calgary, 2500 University Drive NW, Calgary, AB, T3A 5S5, Canada

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. Scenarios and use cases have become popular for capturing and analyzing requirements. However, little research has been done that compares different approaches and assesses their suitability for the telecommunications domain. This paper defines evaluation criteria and then reviews fifteen scenario notations. In addition, twenty-six approaches for the construction of design models from scenarios are briefly compared.

Keywords: design model, requirements, scenario, synthesis, telecommunications, use case

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 [Ardis et al., 13; Hodges and Visser, 48]. 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 [Hurlbut, 51, 52; Jarke and Kurki-Suonio, 55; Weidenhaupt et al., 107]. 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 require-

ments 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.

For several years, scenarios have been very popular in the telecommunications domain. Telecommunication systems consist of many distributed components that interact through information flows. Over the last 150 years, the global telecommunications network has become the largest, distributed network worldwide, and is characterized by a tremendous heterogeneity. It contains components with different functionalities, age, and standards, which are located in different countries and other types of administrative domains. Despite this diversity, these components have to interwork in a reliable manner. The distributed, interactive and incremental nature of telecommunication systems makes scenario notations one of the best means of modeling the interactions between the various components. In fact, one of the first scenario notations, the so-called Message Sequence Charts (MSCs), has its roots in the telecommunication domain and has since then been standardized by the International Telecommunications Union (ITU) as Z.120 [72]. Over the years, scenario notations have become a state-of-the-art modeling tool in telecommunication service development.

Because of the tremendous 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. This paper serves different purposes:

- To give an overview of typical scenario notations applicable to the telecommunication domain. Fifteen such notations are briefly reviewed in section 1.
- To provide means of comparing and quickly evaluating scenario notations based on several criteria relevant to the development of telecommunication systems. These criteria are also introduced in section 1, and they are used to compare the selected scenario notations. Table 2, which summarizes this evaluation, can be used as an index to determine notations of interest, which can vary depending on the target development phase (requirements, design, implementation, testing, etc.).
- To give an overview and brief comparison of selected analytic and synthetic approaches for the construction of detailed models (e.g., analysis, designs and implementations) from scenarios (section 2). Table 4 provides an evaluation summary which can be also used as an index for finding relevant techniques based on particular source scenario notations or on target modeling languages.
- To discuss challenges and hopes for the future, with a special emphasis on ITU-T's upcoming User Requirements Notation (URN) and on the Unified Modeling Language (UML) 2.0 (sections 3 and 4).

- To provide readers with an extensive and current bibliography.

Given the scope of this study and the number of surveyed notations and techniques, this paper can neither offer illustrative tutorials or examples, nor can it provide an empirical comparison based on a common case study. However, we believe that this paper will still provide readers with knowledge useful for making informed decisions about the suitability of particular notations and techniques in their development process.

1. Scenario notations

1.1. 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 community confirmed this trend about a decade ago [Jacobson et al., 54]. 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 systems.

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 [Regnell, 80]. There is no agreed distinction 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) [103]. 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 [103; Regnell et al., 81]. 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 [van Lamsweerde and Willemet, 106; Rolland et al., 84]. 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 [Benner et al., 17] and in research on inquiry-based requirements engineering [Potts et al., 78].

The use of scenarios for requirements/software engineering bears benefits and drawbacks. A non-exhaustive list of the most relevant ones follows in table 1. These benefits and drawbacks are not exclusive to the telecommunications domain, but telecommunications specifics are sometime emphasized.

The increasing popularity of scenarios suggests that their benefits outweigh their drawbacks. Further, scenario notations do not have to be used in isolation, and combin-

Table 1
Benefits and drawbacks of scenarios.

Benefits	Drawbacks
<ul style="list-style-type: none"> ● 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 (which are critical in many reactive systems and telecommunication systems). ● They can be introduced in iterative and incremental design processes (which are frequent when dealing with new telecommunication features or services). ● They can abstract from the underlying system structure, if necessary (e.g., when developing a platform-independent standard). ● They are most useful for documentation and communication. ● They can guide the requirements-based tests generation for validation at different levels (specification, design and implementation). ● They can guide the construction of more detailed models and implementations (this theme is revisited in section 2). 	<ul style="list-style-type: none"> ● 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 (e.g., when produced by different design teams or contributors to standards). ● Scenarios are often unable to express many non-functional requirements (such as robustness, reliability, etc.) other than by giving examples or instances. ● Scenarios often leave required properties about the intended system implicit. ● The synthesis of components behavior, from a collection of scenarios, remains a complex problem. ● The use of scenarios leads to the usual problems related to traceability with other models used in the development process. ● Getting and maintaining the right granularity for the scenarios can be a real challenge. ● Design approaches based on scenarios are rather recent and seldom possess a high level of maturity. Scalability and maintainability represent notably important issues (especially given the complexity of telecommunication systems).

ing them with other languages or techniques can sometimes cure several of the drawbacks identified. For instance, many construction techniques presented in section 2 use formal target languages to enable completeness and consistency/granularity checking. Combining scenarios with goal-oriented languages also help expressing non-functional requirements.

1.2. Evaluation criteria

The following collection of nine criteria will be used to categorize and compare many scenario notations relevant to the development of telecommunication systems and services. Some of these criteria (e.g., decomposition and abstraction) are domain-independent and focus on scenario aspects essential to any software/system development process. Others focus on issues of particular importance to the proper modeling of telecommunication services (e.g., component focus and ordering). Both aspects are necessary and often they overlap.

Component focus. Scenarios can be described in terms of communication events between system components (i.e. *component-centered*), or else independently from components, in a pure functional style (i.e. *component-independent*). 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. Many telecommunication services and standards can first be described independently from an underlying architecture (which is often vendor-specific or technology-specific). An early focus on messages at the functional requirements level may lead to system overspecification, may prune out other appropriate options, and may introduce unnecessary difficulties when evolving systems and services towards different infrastructures and product families.

Hiding. Scenarios could describe system behavior with respect to their environment only (black-box), or they could include internal (hidden) information as well (gray-box). According to Chandrasekaran, the most important reason that impeded the progress of various large projects he studied is the lack of internal details in scenarios [Chandrasekaran, 23]. 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 [Zave and Jackson, 112]. 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, which are often stringent in telecommunication systems (e.g., performance and fault-tolerance). 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, logic, grammars, trees, state machines, tables, visual paths, and sequence diagrams. Graphical representations are often better understood by a wide range of stakeholders (especially telecommunication engineers and standardization bodies), 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 that can be ordered *sequentially* or *causally*. Causal ordering is very important when concurrency is involved (as it is the case in most telecommunication systems due to multiple threads or distributed implementations), otherwise concurrent actions expressed with a sequential ordering might re-

sult in logical fallacies and other incorrect assumptions at the requirements level. Causal ordering also includes sequential ordering.

Time. Support for expressing time constraints and timeout mechanisms is essential to several telecommunication applications (e.g., multimedia services). Different notations support time to different degrees that we qualify as *none*, *partial*, and *full* (i.e. with appropriate data types and evaluation mechanisms).

Decomposition. Due to the complexity of telecommunication systems, it is often necessary to decompose lengthy scenarios into smaller, reusable pieces. Decomposition in a scenario notation can be *hierarchical* (which improves scalability) or be achieved through *dependencies* (e.g., references, contains, etc.). Dependencies are usually more flexible in nature but less prone to formal analysis. *References* to other scenarios are considered here as partial support for decomposition.

Abstraction. An *abstract* scenario is generic, with formal parameters, whereas a *concrete* scenario focuses on one specific instance, with concrete data 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. Abstraction is beneficial in telecommunication systems because many data parameters are usually involved, and hence related concrete scenarios can be regrouped. A scenario notation can also support *both* abstract and concrete scenarios.

Identity. Scenarios can focus on *one actor* or target *many actors* at once. The latter view improves the expressiveness of notations when describing end-to-end situations whereas the former view is useful for component-oriented implementations.

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, Web services, and negotiation mechanisms can benefit from notations that can express dynamicity.

Other criteria such as *multiplicity* (where multiple related traces can be connected as one scenario) and *modality* (acceptance or rejection scenario) are not really helpful in distinguishing scenario notations because most notations support these criteria. Other sets of criteria have also been defined in the literature. For instance, Cockburn uses four dimensions to use case descriptions, namely purpose, content, plurality, and structure [Cockburn, 26]. Purpose can be either for stories (explanations) or for requirements. Content can be contradicting, consistent prose, or formal content. Plurality is either 1 or multiple, in a way similar to multiplicity. Structure can be unstructured, semi-formal, or formal. This dimension shares some common characteristics with our representation criterion. Rolland et al. propose another classification based on four views (form, content, purpose, lifecycle), which are composed of more fine-grained criteria [Rolland et

al., 84]. The first two views are very much in line with many of the criteria introduced in this section, whereas the last two views relate to the usage of scenarios in development activities (outside the scope of our study). In a European industrial survey, Arnold et al. further extended these four views to produce a classification taxonomy for scenarios usage in industrial projects [Arnold et al., 14]. Their criteria are grouped under five main divisions: project properties, scenario contents and representation, goals, process, and experiences and expectations. They surveyed twelve industrial projects from various domains (telecommunications, sales, medical, software development, insurance, banking) where scenarios are used. Their second division is where our criteria focus, but we offer a stronger emphasis on telecommunications aspects.

Given their nature, some scenario notations are more suitable than others for particular development stages. For instance, abstract and component-independent notations will be more useful in early requirements activities whereas concrete and component-centered notations will be more suitable for detailed design and for testing. Hence, there is no single notation that can cover all stages of a development cycle properly, and a complete development methodology may involve multiple scenario notations.

1.3. Selected notations

There are dozens of scenario notations used for the description of system usage, goals, and business logic. Hurlbut's thesis surveyed nearly sixty different scenario, use case, and policy formalisms and models, and others have emerged since then [Hurlbut, 51, 52]. This section focuses on a selection of fifteen scenario notations particularly relevant to the telecommunications domain (some of which are briefly illustrated in figure 1), and it provides a concise comparison in terms of the criteria introduced in section 1.2. These notations were selected based on the experience of the authors with many collaborative industrial and standardization projects. Other scenario notations may be applicable to the telecommunications domain (including proprietary notations), but we believe that the main trends are covered by our selection.

Message Sequence Charts (MSCs). The scenario notation that is the most commonly used by telecommunications companies and standards bodies is undoubtedly message sequence charts [72]. 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 abstract or concrete events (messages), and centered towards components. MSCs can represent internal actions and multiple actors. While basic MSCs mostly focus on simple traces, High-level MSCs (HMSCs) enable the structuring and hierarchical decomposition of scenarios. Causality is supported at both levels through inline expressions, coregions, and parallel composition. MSCs also include good support for timers and various time constraints. 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 [Kimbler and Søbirk, 57; Regnell et al., 81]. Andersson and

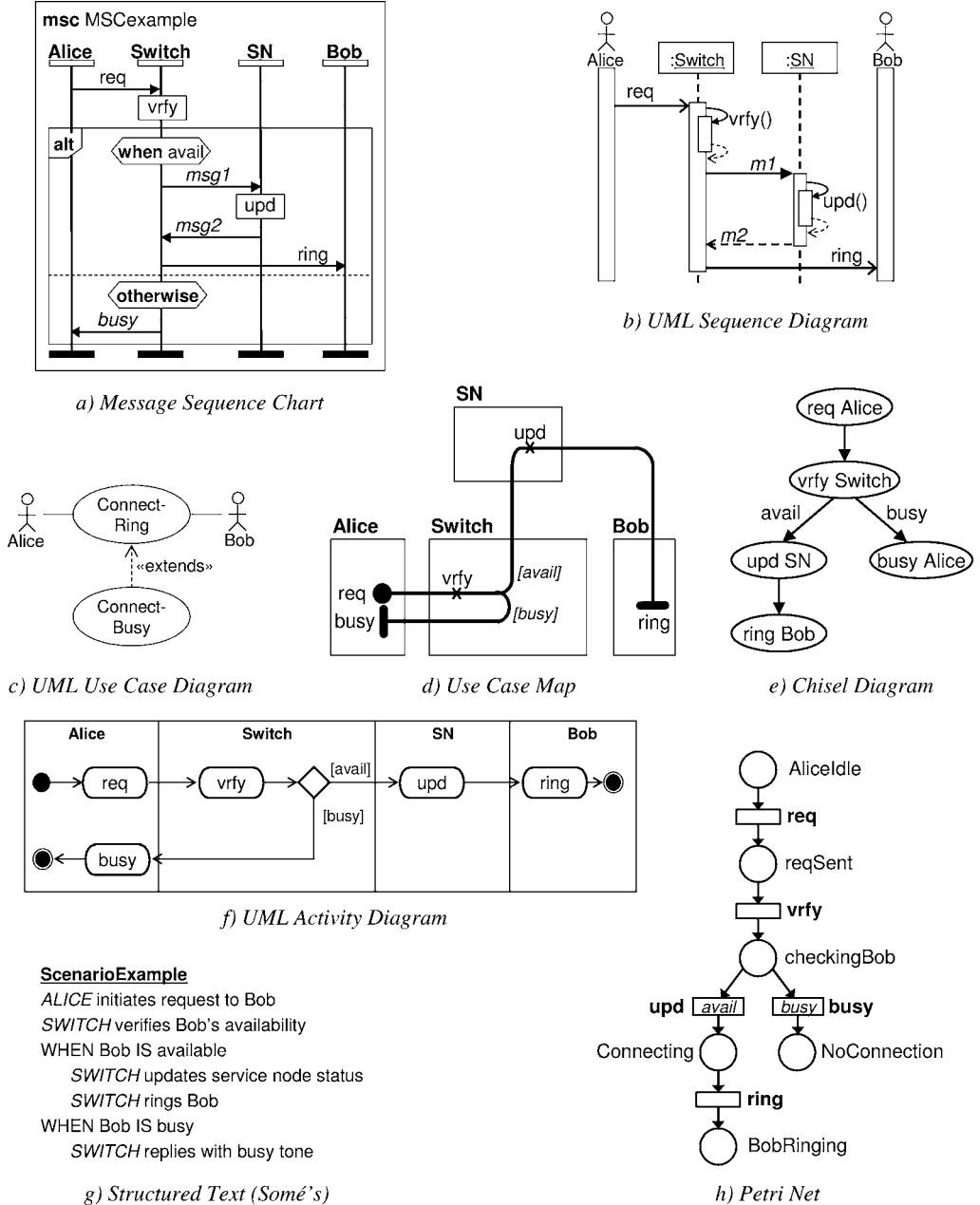


Figure 1. Illustration of several scenario notations.

Bergstrand also present a method to formalize use cases that introduces an unambiguous syntax via MSCs [Andersson and Bergstrand, 12].

Use cases. Jacobson's use cases are prose descriptions of behavior from the user's perspective [Jacobson et al., 54]. They are mostly black-box, i.e. they focus on the interactions between actors and systems. Use cases can be of two kinds: basic courses for normal scenarios, and alternative courses, which include fault-handling scenarios. Use cases are based on sequential ordering, they represent abstract scenarios, and they may involve many actors.

UML use case diagrams. These diagrams offer a graphical means by which use cases can be related to each other and to external actors [103]. They offer dependencies such as *uses* and *extends*, which allow for use cases to reuse other scenarios or part thereof (although hierarchical decomposition is not supported as such). Use case diagrams provide a contextual view of who participates in a use case and of various dependencies, but they provide little insight on the precise nature of scenarios (not even the ordering of events) without looking at the use cases themselves.

UML sequence diagrams. Similar to MSCs, these diagrams describe patterns of interaction among objects, but in a more limited way [103]. UML sequence diagrams are sequential in nature, without decomposition mechanism, and without specific support for time. The same information can be represented in another form called *collaboration diagrams*, which show the system structure (in two dimensions) and the component interactions (numbered sequentially) involved in a scenario. UML sequence diagrams are often used to formalize use cases in use case diagrams.

CREWS-l'écritoire. 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 [Ben Achour et al., 16]. This notation is supported by a tool called *L'écritoire* [Rolland et al., 84] and, to some extent, by the SAVRE tool [Maiden, 68]. 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 is sequential. Decomposition and structuring can be achieved through connections to AND-OR goal networks. Leite et al. [63] offer an interesting alternative to CREWS' scenarios (structured textual language, with hierarchical decomposition based on a bottom-up approach) but to our knowledge it has not been used in a telecommunication context.

Scenario trees. Hsia et al. [50] suggest the use of scenario trees that represent all scenarios for a particular user. 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 [18] proposes use case trees as a text-based notation for describing scenarios related to one entity. This notation, inspired from the TTCN-2 testing notation [76], 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 causality, multiple entities and hiding limits its usefulness as a requirements notation.

Chisel diagrams. Aho et al. [2] have performed empirical studies with telecommunication engineers to create the chisel notation. 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. Decomposition is partially supported through references, and there is no language construct for the explicit support of time.

Statechart diagrams. Glinz [37] uses Harel's statechart notation [Harel and Gery, 41], now part of UML [103], as a way of capturing scenarios. 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 through sequence, alternative, iteration, or concurrency declarations. These scenarios support causal ordering, multiple actors, and multiple abstract scenario sequences.

Live Sequence Charts (LSCs). Damm and Harel [27] propose LSCs, which enrich a subset of MSCs with a concept called *liveness*. Through the same representation, their formal support for liveness enables one to specify forbidden scenarios (e.g., to capture safety requirements) and to distinguish between mandatory and optional scenarios. This can lead to more accurate requirements, component descriptions, and test cases. Like MSCs, LSCs support decomposition (through subcharts), causality, and time (through

timers and clock variables). LSCs do not have a higher-level view like HMSC, but such a view (high-level LSCs) is proposed by [Bontemps and Heymans, 19]. Concrete values can be used in scenarios, and symbolic values also provide an appealing support for abstract scenarios. Dynamicity is in a way supported by symbolic messages and instances, and by the dynamic creation and destruction of symbolic instances.

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 [Somé et al., 93; Somé, 92]. 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 a previous version of the MSC notation (MSC 1992) to support additional scenario elements such as conditions and expiration delays (now covered to some extent by the MSC 2000 standard). Multiple abstract scenarios and actors can be considered by these component-based notations. They are ordered sequentially, although nonlinear causality appears when composing the scenarios together to form an automaton.

RATS. In the RATS (*Requirements Acquisition and specification for Telecommunication Services*) methodology, Eberlein [34] uses three different scenario representations: textual (natural language), structured (in text, but with pre/post/flow conditions) and formalized (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 focus first 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.

Petri Nets (PNs). Petri nets, which have existed for decades [Reisig and Rozenberg, 82], are abstract machines used to describe system behavior visually with a directed graph containing two types of nodes: places and transitions. Places, represented by circles, contain tokens whereas transitions, represented by lines or rectangles, allow tokens to move between places. An event usually corresponds to the firing of a transition, which is allowed when all arrows entering the transition originate from places with tokens. PNs can be used to capture sequential, alternative, and concurrent scenarios in a component-independent way. PNs with data types are now being standardized by ISO/IEC as *high-level Petri nets* [47], which will include a textual format described in SGML. Although the first version of the standard will not support time and decomposition explicitly, future versions will likely integrate language features found in various PN extensions such as object coloured PNs (dynamicity, class nets and inheritance) or hierarchical CPNs (subnets).

Use Case Maps (UCMs). The use case map notation is used for describing *causal relationships* between *responsibilities*, which may optionally be bound to underlying organizational structures of abstract *components* [Buhr, 22; 101]. 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 encapsulate 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 would 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, and they have partial support for time constructs with timers and time-out paths. Concrete scenarios can be extracted using a simple path data model (Boolean variables) and *scenario definitions*, where initial values and triggered start points are provided.

UML activity diagrams [103]. This type of UML behavioral diagram 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 and can be component independent. 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, they do not support time constructs, and the binding of actions to “components” is semantically weak in the current UML standard.

1.4. Summary of evaluation

The fifteen scenario notations compared in this paper are summarized in table 2.

MSCs, LSCs, and UML sequence diagrams are very 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) during the development of detailed requirements, designs, and test cases. Unlike (H)MSCs and LSCs, UML sequence diagrams do not really support decomposition, causality, and time. LSCs promise interesting benefits over the more established MSC notation (e.g., liveness and some dynamicity). Use cases and UCTs are generally not used to describe internal responsibilities and they do not support causal ordering. UML use case diagrams provide some structuring useful as an overview mechanism, but their lack of ordering makes them incapable of expressing operational scenarios on their own. 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 limits their usefulness for distributed systems composed of communicating entities. Chisel diagrams represent a good alternative to scenario trees, but they still focus on components interactions and lack support for time and hiding. Somé's scenarios lack first-class causal ordering, which only appears when scenarios are transformed into component automata. Given their nature, statecharts tend to bring the focus too quickly on component behavior.

RATS, Petri nets, UCMs, and UML activity diagrams support component-independent scenarios, which are useful for early descriptions of requirements and help avoiding early commitments to a specific architecture. Unlike the other notations, the upcoming Petri net standard does not support the allocation of scenarios to components (useful when moving towards the design phase), and it does not support decomposition. Where RATS use structure text, UCMs and UML activity diagrams show graphically the allocation of scenario elements to components. However, the UCM representation uses 2-D architectural view (where components can contain subcomponents) whereas UML activity diagrams are limited to vertical swimlanes, which are less evocative (compare diagrams d and f in figure 1). UCMs have partial support for time constructs and they can also capture dynamicity through dynamic stubs (with multiple submaps selected at run-time) and dynamic responsibilities (which can move components and submaps around and store them in pools). 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) [94] and ISO's LOTOS (Language Of Temporal Ordering Specification) [53]. Such models are usually executable and suitable for validation, verification, test derivation, code generation, and understanding/evolution of legacy applications.

2. Construction approaches

2.1. 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 [Probert and Saleh, 79; Saleh, 86, 87], Saleh and Probert present two categories of construction

Table 2
Evaluation of the selected scenario notations.

Scenario notation	Comp.-cent./ Comp.-indep.	Hiding	Represent- tation	Ordering	Time	Decompo- sition	Abstraction	Identity	Dynamicity
MSC	C-C	Yes	Sequence diagram	Causal	Full	Hierarchical	Both	Many	Static
Use case	C-C	No	(structured) text	Sequence	None	References	Abstract	Many	Static
UML use case diag.	C-I	Yes	Graph	None	None	Dependencies	Abstract	Many	Static
UML seq. diagrams	C-C	Yes	Sequence diagram	Sequence	None	None	Both	Many	Static
CREWS'	C-C	No	Structured text	Sequence	None	References	Abstract	Many	Static
Scen. tree	C-C	No	Tree & grammar	Sequence	None	None	Abstract	One	Static
UCT	C-C	No	Text & grammar	Sequence	Partial	Hierarchical	Concrete	One	Static
Chisel	C-C	No	Tree	Causal	None	References	Abstract	Many	Static
Statechart	C-C	No	State machine	Causal	None	Hierarchical	Abstract	Many	Static
LSC	C-C	Yes	Sequence diagram	Causal	Full	Hierarchical	Both	Many	Dynamic
Some's	C-C	Yes	Structured text & sequence diagram	Sequence	Partial	None	Abstract	Many	Static
RATS	Either	Yes	Structured text	Causal	None	Hierarchical	Both	Many	Static
Petri nets	C-I	No	Graph	Causal	None	None	Abstract	Many	Static
UCM	Either	Yes	Paths on 2-D components	Causal	Partial	Hierarchical	Abstract	Many	Dynamic
UML act. diagrams	Either	Yes	Paths on linear swimlanes	Causal	None	Hierarchical	Abstract	Many	Static

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 effects 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 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 other benefits and drawbacks, many of which are summarized in table 3.

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 telecommunication systems in general.

2.2. 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, twenty-six of which are reviewed next. Additional evaluation criteria for the selection of a suitable technique in a given context include:

Table 3
Benefits and drawbacks of construction approaches.

	Benefits	Drawbacks
Analytic	<ul style="list-style-type: none"> • No formal source model required. • Both the source and target models can exploit the richness and expressiveness of their respective modeling language to their full extent. • 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. • Non-functional requirements (e.g., performance, robustness) can more easily be taken into consideration. 	<ul style="list-style-type: none"> • Transformation mostly manual. • Errors may result from the construction. • Verification is required. • Many iterations may be required to fix the errors detected during verification. • Time-consuming.
Synthetic, interactive	<ul style="list-style-type: none"> • Precise algorithmic transformation. • Improper synthetic constructions can be avoided by interacting with the designer. • Correctness “ensured” by construction (under certain assumptions). Many faults are therefore avoided. • Verification theoretically not required. • Only one iteration required. • Quick construction. 	<ul style="list-style-type: none"> • Not fully automated. • Requires formal and detailed source models. • May require a partially constructed model to be available. • Both source and target modeling languages are usually restricted in style and content. • Difficult to take into consideration design/implementation constraints, optimizations, and non-functional requirements. • Resulting model usually hard to understand, maintain and extend.
Synthetic, automated	<ul style="list-style-type: none"> • Precise algorithmic transformation. • Fully automated. • Correctness “ensured” by construction (under certain assumptions). Many faults are therefore avoided. • Verification theoretically not required (for certain properties). • Only one iteration required. • Very quick construction. 	<ul style="list-style-type: none"> • Requires formal source models. • Both source and target modeling languages are usually restricted in style and content. • May result in improper synthetic constructions in ambiguous cases (the algorithm makes the decision, not the designer). • Requires more details in the source model than non-automated approaches. • Resulting model often hard to understand, maintain and extend.

- Type of construction approach: analytic, synthetic non-automated (includes interactive synthesis), or synthetic automated. The benefits of each option were presented in table 3.
- Source scenario notation, such as the ones described in section 1.3. Previous investments in a particular notation, with associated tools and expertise, is an important factor to consider when selecting a suitable approach.
- Whether the scenario model requires explicit components and messages. This can influence the choice of the development phase (e.g., requirements or design) where the notation should be used.
- Target construction model (SDL, UML statecharts, automata, LOTOS, etc.). This obviously will affect the potential usages of the target model (e.g., analysis or code generation). Again, previous investments in a particular language may restrict users to compatible approaches.

On various occasions, tool support will also be discussed informally.

2.3. Selected approaches

This section focuses on twenty-six construction approaches particularly relevant to the telecommunication domain, and it provides a concise comparison in terms of the criteria seen in section 2.2. Again, these techniques were selected by the authors based on previous experiments with collaborative industrial and standardization projects, and on results from various conferences and workshops on scenarios.

Non-automated analytic approaches

The Usage Oriented Requirements Engineering (UORE) approach proposed by Regnell et al. [Regnell et al., 81; Regnell, 80] builds on the Objectory methodology [Jacobson et al., 54] 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, testing, and validation activities. No automated support is provided.

In RATS, Eberlein [34] provides informal guidelines. 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 here 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 in depth

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) [Bordeleau and Buhr, 21], which combines the use of scenario textual descriptions (use cases), UCMs, MSCs, and ROOM (now UML-RT) [Selic et al., 91]. Included is an approach where UCM scenarios are first transformed into HMSCs, and then into hierarchical communicating finite state machines (ROOMCHARTS) [Bordeleau, 20]. No construction algorithm is proposed, but the use of transformation patterns is suggested instead. Several such patterns are provided for the UCM-HMSC mapping, 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. [62] present a related technique for the transformation of a set of MSCs to a statechart model, 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 (SPEC-VALUE) [Amyot et al., 7; Amyot and Logrippo, 9; Amyot, 6], where UCMs describe functional scenarios optionally bound to architectural components, and where LOTOS [53] formalizes the integration of scenarios and (if needed) 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. 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 [van Lamsweerde and Willemet, 106], a drawback of scenarios is that system properties are often left implicit. If these properties were explicit (e.g., in declarative terms), then consistency and 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. Their scenarios are sequential and synchronous 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.

Heymans and Dubois [46] offer rules, heuristics, and templates that enable the construction of *Albert II* specifications from *Action Sequence Charts* (ASCs). The ASC scenario notation is a subset of the MSC language with several extensions to model causality (with actions, reactions, and composition) and semantics in line with the *Albert II* language. *Albert II* is a formal language based on real-time temporal logic which supports declarative and operational descriptions of functionalities [Du Bois, 32]. Although the source scenario notation requires components, the target model is global, hence addressing C1. *Albert II* specifications are mathematical formulae, which are sometimes difficult to read. This motivates the need for a more informal and intuitive source notation (scenarios) and for an animator. The *Albert II* animator tool presented in [Heymans, 45] supports model validation (e.g., against the ASCs) and exploration.

Yee and Woodside [111] have developed a transformational approach to process partitioning using timed Petri nets, which addresses C2. An abstract scenario model combining both the system and its environment (*Process Specification of Requirements* – PSR) is partitioned, using a collection of correctness preserving transformations (abstraction, refinement, sequentialization, partitioning, and resource access control), into a collection of communicating processes that can represent system components (called proto-design). Both the source and the target models are described using timed Petri net, and the transformations ensure their behavioral equivalence from the environment viewpoint. Being executable, the target model can be used for analysis and for performance evaluation of alternative architectures. The source model does not require any component, but the selection and application of the transformations are manual.

Dano et al. [29] use Petri nets as an intermediate formalism in their construction approach. Domain experts first produce scenarios using a tabular/textual notation, where the states of the objects involved can be shown. Scenarios are sequential, but alternatives can be captured. Objects are not required to be identified at first, but the target model can be component-based (C2) once they are introduced. Scenarios are formalized as synchronized coloured Petri nets through the application of mapping rules. The formalized scenarios are then integrated using seven types of temporal links which define how two scenarios overlap (or not) over time. Another set of transformation rules is used to generate the target (OMT) state transition diagrams for each object type.

Chen and Ural [24] present rules used to construct deadlock-free designs and communication protocols from sequential scenarios called *observations*, which focus on transmissions and receptions of messages. Since the suggested application of this approach is the reverse-engineering of process behavior from execution histories, there is no concrete representation of the source scenario model, but common subsets of MSCs and UML sequence diagrams can be used in a forward-engineering context. The target model is a set of communicating finite state machines (CFSMs) connected through FIFO queues (C2). The number of components is fixed. The rules aim the avoidance of deadlocks and unspecified receptions in the CFSM model, as well as the reduction of the

number of states. They guarantee the absence of deadlocks in models with two entities only.

Non-automated synthesis approaches

Desharnais et al. [31] propose a synthesis approach for the integration of sequential scenarios represented in state-based relational algebra. 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 presents a composition algorithm that transforms his scenarios into Alur's timed automata [Alur and Dill, 3], one for each component (hence addressing C2) [Somé et al., 93; Somé, 92]. 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. One original point to notice is that 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 with the scenarios, which is seldom the case of other (semi-automated) synthesis techniques. This work was further extended by Salah et al. with improved support for automation [Salah et al., 85].

Harel and Kugler propose an algorithm for the synthesis of statecharts from a subset of the Live Sequence Chart (LSC) notation, without data or conditions [Harel, 40; Harel and Kugler, 42]. This algorithm decides the satisfiability and consistency of a set of LSCs, something that is harder to do than for MSCs due to the possibility of expressing mandatory and forbidden scenarios. The proof of consistency produces a global system automaton (which can be quite large). 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. However, [Damm and Klose, 28] developed a verification tool (integrated to the STATEMATE tool) where LSCs are used to test a statechart model. Bontemps and Heymans [19] have also initiated work towards the synthesis of Büchi automata from (high-level) LSCs. However, the resulting automaton is global (hence addressing C1) and they are working towards refining their work to generate one automaton per instance.

Alur et al. have an algorithm that transforms a set of stateless basic MSCs into communicating state machines of various types (C2) [Alur et al., 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 deadlocks are automatically avoided under some conditions), 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 [Alur et al., 5], who extended this work in another

direction to support HMSCs during requirements analysis with the tool uBET [Holzmann et al., 49].

Mäkinen and Systä [69] have developed an approach and tool to synthesize UML statechart diagrams from a set of UML sequence diagrams [Systä, 95], hence addressing C2. Since fully automated synthesis may overgeneralize the statechart and may introduce more scenarios than described in the sequence diagrams, the MAS (*Minimally Adequate Synthesizer*) approach is interactive. MAS models the synthesis process as a language inference problem and translates sequence diagrams first into traces, then into finite state automata, and finally into statechart diagrams. The interactive part of the tool asks membership queries visualized as sequence diagrams (in a nutshell: “Is this sequence diagram acceptable?”), which allow the derivation of a consistent and deterministic statechart diagram. Counter-examples can be provided when appropriate. This work was extended by Koskinen et al. [61] to support inaccurate answers during interactive sessions (probably yes, probably no, I do not know, etc.).

Automated synthesis approaches

With their SCED methodology [Koskimies and Mäkinen, 59], 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 diagrams, which are based on Harel’s statecharts. The synthesis is supported by the SCED tool [Koskimies et al., 60], 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 [Schönberger et al., 89], 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.) [Elkoutbi, 36].

Whittle and Schumann [108] propose an algorithm for the generation of UML statecharts from a collection of UML sequence diagrams (C2). It allows for conflicts to be detected and resolved through UML’s *Object Constraint Language* (OCL) and global state variables. These statecharts can be nondeterministic. 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 et

al., 89; Somé et al., 93] 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 linked to the *MagicDraw* UML environment. In a recent application to an air control system, the authors adapted their work towards the support of existential, universal, and generalized scenarios, more in line with LSCs [Whittle and Schumann, 109].

Uchitel and Kramer [99] address C2 with a synthesis algorithm that transforms MSCs into Finite Sequential Processes (simple process algebra) [Magee and Kramer, 67]. The source MSC semantics is given in terms of labeled transition systems with parallel composition, and HMSCs are used to structure (sequential) MSCs. State labels on the MSCs indicate component states. The authors claim this helps taking into consideration domain-specific assumptions (e.g., when should states be merged) directly at the MSC level rather than in the synthesis algorithm itself or in complementary languages (like the use of OCL by Whittle and Schumann). Their algorithm is implemented in Java, and the resulting finite sequential processes can be fed to an analyzer for model checking and animation [Magee and Kramer, 67; Uchitel et al., 100].

Leue et al. have developed two algorithms for the automated synthesis of Real-Time Object-Oriented Modeling (ROOM) models from standard HMSC scenarios [Leue et al., 64]. 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 [Ben-Abdallah and Leue, 15], which also supports Promela as target languages. Their authors claim that their work can be adapted to support SDL and UML.

Mansurov and Zhukov [71] address C2 and target the automated generation of SDL models from HMSCs. 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. SDL object types are generated for type-based entities (roles) in MSCs. The resulting SDL system usually allows more traces than those defined by the HMSCs and, although the algorithm attempts not to add anything wrong, deadlocks may also result from composing incoherent scenarios. Several constraints on the source HMSCs must be satisfied (e.g., no inline statements, no parallel constructs, and high level of detail). This technique is implemented in MOST, the *Moscow Synthesizer Tool*, in now commercialized by KLOCwork and has recently been integrated to Tau 4.4 (Telelogic's SDL tool) as the MSC2SDL synthesizer. An application of this technique to the reverse-engineering and evolution of telecommunication software is illustrated in [Mansurov and Probert, 70].

Li and Horgan [65] 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*. 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 others, hence they should be consistent and have no undesirable interaction, which is of limited use for early validation. Nondeterminism 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 [56] 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. 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 [Abdalla et al., 1], and then merge the behavior descriptions of the old SDL with the increment SDL, on a per process basis. If nondeterminism 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 [98] 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. 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.

Guan [39] provides a synthesizer for the generation of LOTOS models (useful for exploration and formal validation) from use case map scenarios. Her work automates many of the construction rules proposed in Amyot's SPEC-VALUE [Amyot, 6] as well as others in a Java tool called *Ucm2LotosSpec*, which uses as input the XML files produced by the UCM Navigator tool [Miga et al., 75]. Input scenarios may or may not be allocated to components, but if they are then the resulting model becomes component-based (C2). These scenarios must be expressed using a subset of the UCM notation

(timers, aborts, explicit loops, dynamic components, dynamic responsibilities, and the UCM path data model are not supported). When a scenario traverses many components, messages are created automatically to preserve causality across components and enumerated data types are generated accordingly in the LOTOS model. The resulting specification can be used for formal analysis and animation.

Dulz et al. [33] present an approach where performance prediction models (in SDL) are also automatically synthesized from MSC scenarios, but this time supplemented with performance annotations. 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 [Woodside et al., 110]). 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.

2.4. Summary of evaluation

Several aspects of the construction approaches reviewed in section 2.3 are summarized in table 4. These aspects correspond to the evaluation criteria specified in section 2.2.

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 (H)MSCs, various extensions to MSCs, LSCs, and UML sequence diagrams are commonly used as source scenario models for construction approaches. Techniques based on HMSCs can further benefit from theoretical results on necessary conditions for the synthesis of communicating automata from HMSCs [Hélouët and Jard, 44]. For target construction models, communicating finite state machines, whether they are hierarchical ((UML) statecharts, ROOMCHARTS, or OMT state diagrams) or not (SDL or plain CFSMs) are very common. Algebraic languages (e.g., Albert II, KAOS, LOTOS, finite sequential processes) represent another popular family of target languages. Most techniques addressing construction level C1 produce global target models.

It is difficult to determine the best construction approach for component-based scenarios as they use many different combinations of source and target models. Most techniques are still immature, require more industrial-scale experimentation, and are not supported by industrial-strength commercial tools.¹ Synthesis approaches also have different sets of constraints and design decisions embedded in their algorithms, or different ways of letting users guide such decisions (e.g., use of OCL, state labels, or interactive queries).

Only six of the techniques surveyed do not start from scenarios expressed in terms of components and messages, and five of them are analytic construction approaches.

¹ <http://www-i2.informatik.rwth-aachen.de/Research/AG/MCS/MSC/> discusses many tools related to MSCs.

Table 4
Evaluation of the selected construction approaches.

Construction approach	Level C1/C2	Type of approach	Scenario model	Comp.-based	Construction model
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
Heymans and Dubois	C1	Analytic	Action sequence charts	Y	Albert II
Yee and Woodside	C2	Analytic	Timed Petri nets	N	Timed Petri nets
Dano et al.	C2	Analytic	Tabular/textual	N	State transition diagrams (OMT)
Chen and Ural	C2	Analytic	Sequential sequence diagrams	Y	CFSMs
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
Mäkinen and Systä (MAS)	C2	Synthetic, non-automated	UML sequence diagrams	Y	UML statecharts
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
Uchitel and Kramer	C2	Synthetic, automated	HMSCs	Y	Finite sequential processes
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
Guan (SPEC-VALUE)	C2	Synthetic	UCMs	N	LOTOS
Dulz et al.	C1	Synthetic, automated	Extended MSCs	Y	SDL

Half use tables or Petri nets whereas the other half starts from use case maps. These source scenario notations can abstract from component communication aspects, they are less coupled to a specific architecture, and they are closer to being pure operational 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.

An interesting characteristic of the UCM and LOTOS languages is that they can both address C1 and C2, with or without components. 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 evolved accordingly.

3. 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 software engineering, OO/UML, requirements engineering, and formal methods communities (e.g., [Egyed et al., 35; Systä et al., 96]). 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 plans for UML 2.0 at the OMG include an improved semantic basis for activity diagrams and the integration of most features of message sequence charts (to replace the current sequence diagrams) [102]. UML 2.0 should be available in 2003. Additionally, the OMG and ITU-T Study Group 17 are working towards the integration of their scenario-based approaches and notations [30]. UML 2.0 profiles for SDL, MSC, TTCN, URN and others are planned to appear in 2003 and 2004. Hopefully, this will lead to better and more unified approaches for the synthesis of component-based models (SDL, statecharts, etc.) from standard scenario notations. Such approaches could improve the penetration and acceptance of formal methods and of new requirements, design, and analysis techniques. One main challenge however is the reconciliation of all these notations at a semantic level. In particular, one issue is to determine a suitable and common metamodel capturing part of this semantics in order to simplify transformations from one notation to the next.

ITU-T Study Group 17 has also initiated work towards the standardization of a *User Requirements Notation* (URN) [Amyot and Mussbacher, 11; 105], which targets the representation of requirements for future telecommunication systems and services. The current proposal combines two complementary notations: one for business goals and non-functional requirements (NFRs), and a scenario notation for functional requirements. The goal/NFR notation is the Goal-oriented Requirements Language (GRL) [38], which is based in part on the NFR Framework [Chung et al., 25]. NFRs are seldom captured explicitly in design processes (even in UML), and URN represents a first standardization effort towards solving this issue. The scenario notation is Use Case Maps and hence it abstracts from message exchanges [101]. URN fits nicely in the very first stage of existing ITU-T methodologies (such as I.130 [73] and Q.65 [97]) and smoothes the transition towards later stages of development processes. Liu and Yu illustrate the potential of GRL combined with UCMs in [Liu and Yu, 66]. On top of traceability links between goals, NFRs and scenarios, the proposed standard intends to provide additional insights on the derivation of MSCs [Miga et al., 75] and the generation of formal models [Amyot et al., 7; Amyot and Logrippo, 9; Bordeleau, 20; Sales and Probert, 88] and performance models [Petriu and Woodside, 77] from URN models. Links to the UML standard [Amyot and Mussbacher, 10] and tool support are also being investigated [Miga, 74; 104].

The recent integration of important concepts like liveness into scenario notations (as done in LSCs) also offers new hopes for automation. For example, Harel and Marelly envision an environment where users can execute requirement specifications given in LSCs directly, without the need to build or synthesize a system model consisting of statecharts or code [Harel and Marelly, 43]. In a sense, the scenarios combined to the underlying execution engine would become the final implementation itself.

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 standardized construction approaches may eventually become reality. Other major challenges include the integration of scenarios with other types of requirements in requirements management systems, as well as scenario evolution.

4. Conclusions

The development of current and emerging telecommunication systems can benefit today from scenario notations, both at the requirements stage and at the design stage. Due to prolific and enthusiastic researchers worldwide, many scenario-based approaches are now available.

Section 1 focused on fifteen state-of-the-art notations and provided an evaluation against nine criteria relevant to the telecommunications 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 before committing to specific architectures and protocols.

The transition from scenario models to more detailed design and implementation-oriented models is discussed in section 2. Twenty-six 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 remain largely analytic. Further investigation on common case studies would be beneficial in providing more detailed cost-benefit evaluations.

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.

Acknowledgements

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

References

- [1] M.M. Abdalla, F. Khendek and G. Butler, New results on deriving SDL specifications from MSCs, in: *SDL'99, Proc. of the 9th SDL Forum*, Montréal, Canada (Elsevier, Amsterdam, 1999).
- [2] A. Aho, S. Gallagher, N. Griffeth, C. Scheel and D. Swayne, Sculptor with chisel: Requirements engineering for communications services, in: *Fifth Internat. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'98)*, Lund, Sweden, September 1998, eds. K. Kimbler and L.G. Bouma (IOS Press, Amsterdam, 1998) pp. 45–63. <http://www-db.research.bell-labs.com/user/nancyg/sculptor.ps>.
- [3] R. Alur and D. Dill, A theory of timed automata, *Theoretical Computer Science* 126 (1994) 183–235.
- [4] R. Alur, K. Etessami and M. Yannakakis, Inference of message sequence charts, in: *22th Internat. Conf. on Software Engineering (ICSE 2000)*, Limerick, Ireland (ACM, New York, 2000) pp. 304–313.
- [5] R. Alur, G. Holzmann and D. Peled, An analyzer for message sequence charts, *Software Concepts and Tools* 17(2) (1996) 70–77; <http://cm.bell-labs.com/cm/cs/what/ubet/papers/aAfMSCs.ps.gz>.
- [6] D. Amyot, Specification and validation of telecommunications systems with use case maps and LOTOS, Ph.D. thesis, SITE, University of Ottawa, Canada (2001); http://www.UseCaseMaps.org/pub/da_phd.pdf.
- [7] D. Amyot, R. Andrade, L. Logrippo, J. Sincennes and Z. Yi, Formal methods for mobility standards, in: *IEEE 1999 Emerging Technology Symposium on Wireless Communications and Systems*, Richardson, TX, USA, April 1999; <http://www.UseCaseMaps.org/pub/ets99.pdf>.

- [8] D. Amyot, R.J.A. Buhr, T. Gray and L. Logrippo, Use case maps for the capture and validation of distributed systems requirements, in: *RE'99, 4th IEEE Internat. Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, pp. 44–53; <http://www.UseCaseMaps.org/pub/re99.pdf>.
- [9] D. Amyot and L. Logrippo, Use case maps and lotos for the prototyping and validation of a mobile group call system, *LOTOS 23(12)* (2000) 1135–1157; <http://www.UseCaseMaps.org/pub/cc99.pdf>.
- [10] D. Amyot and G. Mussbacher, On the extension of UML with use case maps concepts, in: “*UML*” 2000, *3rd Internat. Conf. on the Unified Modeling Language*, York, UK, October 2000, Lecture Notes in Computer Science, Vol. 1939 (Springer, New York, 2000) pp. 16–31; <http://www.UseCaseMaps.org/pub/uml2000.pdf>.
- [11] D. Amyot and G. Mussbacher, URN: Towards a new standard for the visual description of requirements, in: *3rd SDL and MSC Workshop (SAM'02)*, Aberystwyth, UK, June 2002; <http://www.UseCaseMaps.org/pub/sam02-URN.pdf>.
- [12] M. Andersson and J. Bergstrand, 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/EXJOB/Title_Abstract_Preface.html.
- [13] M.A. Ardis, J.A. Chaves, L.J. Jagadeesan, P. Mataga, C. Puchol, M.G. Staskauskas and J.V. Olnhansen, A framework for evaluating specification methods for reactive systems—experience report, *IEEE Transactions on Software Engineering* 22(6) (1996) 378–389.
- [14] M. Arnold, M. Erdmann, M. Glinz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer and K. Weidenhaupt, Survey on the scenario use in twelve selected industrial projects, Technical Report, Aachener Informatik Berichte (AIB), No. 98-7, RWTH Aachen, Fachgruppe Informatik, Germany (1998).
- [15] H. Ben-Abdallah and S. Leue, MESA: Support for scenario-based design of concurrent systems, Technical Report 97-12, Department of Electrical and Computer Engineering, University of Waterloo, Canada (October 1997); <http://tele.informatik.uni-freiburg.de/Mesa/index.html>.
- [16] C. Ben Achour, C. Rolland, N.A.M. Maiden and C. Souveyet, Guiding use case authoring: Results of an empirical study, in: *RE'99, 4th IEEE Internat. Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, pp. 36–43.
- [17] K.M. Benner, M.S. Feather, W.L. Johnson and L.A. Zorman, Utilizing scenarios in the software development process, in: *Information System Development Process* (Elsevier Science/North-Holland, Amsterdam, 1993) pp. 117–134.
- [18] A. Boni Bangari, A use case driven validation framework and case study, M.Sc. thesis, SITE, University of Ottawa, Ottawa, Canada (1997).
- [19] Y. Bontemps and P. Heymans, Turning high-level live sequence charts into automata, in: *Scenarios and State Machines: Models, Algorithms, and Tools, ICSE 2002 Workshop*, Orlando, USA (2002).
- [20] F. Bordeleau, 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 (1999); http://www.UseCaseMaps.org/pub/fb_phdthesis.pdf.
- [21] F. Bordeleau and R.J.A. Buhr, The UCM-ROOM design method: From use case maps to communicating state machines, in: *Conf. on the Engineering of Computer-Based Systems*, Monterey, USA, March 1997; <http://www.UseCaseMaps.org/pub/UCM-ROOM.pdf>.
- [22] R.J.A. Buhr, Use case maps as architectural entities for complex systems, Special Issue on Scenario Management of *IEEE Transactions on Software Engineering* 24(12) (1998) 1131–1155; <http://www.UseCaseMaps.org/pub/tse98final.pdf>.
- [23] P. Chandrasekaran, 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 Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'97)*, 1997, pp. 6–9.

- [24] X.J. Chen and H. Ural, Construction of deadlock-free designs of communication protocols from observations, *Computer Journal* 45(2) (2002) 162–173.
- [25] L. Chung, B.A. Nixon, E. Yu and J. Mylopoulos, *Non-functional Requirements in Software Engineering* (Kluwer Academic, Dordrecht, 2000).
- [26] A. Cockburn, Structuring use cases with goals, *Journal of Object-Oriented Programming (JOOP/ROAD)* 10(5) (1997) 56–62; <http://members.aol.com/acockburn/papers/usecases.htm>.
- [27] W. Damm and D. Harel, LCSs: Breathing life into message sequence charts, *Formal Methods in System Design* 19(1) (2001).
- [28] K. Damm and J. Klose, Verification of a radio-based signalling system using the STATEMATE verification environment, *Formal Methods in System Design* 19(2) (2001) 121–141.
- [29] B. Dano, H. Briand and F. Barbier, A use case driven requirements engineering process, in: *RE'97, 3rd IEEE Internat. Symposium on Requirements Engineering*, Annapolis, USA (1997).
- [30] Data networks and telecommunication software, ITU-T Study Group 17, Geneva (2002).
- [31] J. Desharnais, M. Frappier, R. Khédri and A. Mili, Integration of sequential scenarios, in: *ESEC'97, 6th European Engineering Conference*, Lecture Notes in Computer Science, Vol. 1301 (Springer, New York, 1997) pp. 310–326.
- [32] P. Du Bois, The Albert II reference manual: Language constructs and informal semantics, Research Report RR-97-002, Computer Science Department, University of Namur, Belgium (July 1997); <ftp://ftp.info.fundp.ac.be/publications/RR/RR-97-002.ps.Z>.
- [33] W. Dulz, S. Gruhl, L. Lambert and M. Söllner, Early performance prediction of SDL/MSD specified systems by automated synthetic code generation, in: *SDL'99, Proc. of the 9th SDL Forum*, Montréal, Canada (Elsevier, Amsterdam, 1999).
- [34] A. Eberlein, Requirements acquisition and specification for telecommunication services, Ph.D. thesis, University of Wales, Swansea, UK (1997); <http://www.enel.ucalgary.ca/People/eberlein/publications/thesis.zip>.
- [35] A. Egyed, T. Systä, S. Uchitel and A. Zündorf, A summary of the ICSE 2002 workshop on scenarios and state machines: Models, algorithms, and tools, *ACM Software Engineering Notes* 27(5) (2002).
- [36] M. Elkoutbi, I. Khriess and R.K. Keller, Generating user interface prototypes from scenarios, in: *RE'99, 4th IEEE Internat. Symposium on Requirements Engineering*, Limerick, Ireland, June 1999, pp. 150–158; <ftp://ftp.iro.umontreal.ca/pub/gelo/Publications/Papers/isre99.pdf>.
- [37] M. Glinz, An integrated formal model of scenarios based on statecharts, in: *Proc. of the 5th European Software Engineering Conf. (ESEC 1995)*, Sitges, Spain, 1995.
- [38] Goal-oriented Requirements Language (GRL), ITU-T, URN Focus Group, Draft Recommendation Z.151, Geneva (2002).
- [39] R. Guan, From requirements to scenarios through specifications: A translation procedure from use case maps to LOTOS, M.Sc. thesis, University of Ottawa, Canada (September 2002); http://lotos.site.uottawa.ca/ftp/pub/Lotos/Theses/rg_msc.pdf.
- [40] D. Harel, From play-in scenarios to code: An achievable dream, in: *Fundamental Approaches to Software Engineering (FASE'2000)*, Lecture Notes in Computer Science, Vol. 1783 (Springer, New York, 2000) pp. 22–34; http://www.wisdom.weizmann.ac.il:81/Dienst/UI/2.0/Describe/ncstrl.weizmann_il/MCS00-06.
- [41] D. Harel and E. Gery, Executable object modeling with statecharts, in: *Proc. of the 18th Internat. Conf. on Software Engineering*, Berlin, March 1996 (IEEE Press, New York, 1996) pp. 246–257.
- [42] D. Harel and H. Kugler, Synthesizing state-based object systems from LSC specifications, *Internat. Journal of Foundations of Computer Science* 13(1) (2002) 5–51; also in: *5th Internat. Conf. on Implementation and Application of Automata (CIAA 2000)*, Lecture Notes in Computer Science (Springer, New York, 2000).
- [43] D. Harel and R. Marelly, *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine* (Springer, Berlin, 2003).

- [44] L. Hélouët and C. Jard, Conditions for synthesis of communicating automata from HMSCs, in: *5th Internat. Workshop on Formal Methods for Industrial Critical Systems*, Berlin, April 2000; <http://www.fokus.gmd.de/research/cc/tip/fmics/abstracts/helouet.html>.
- [45] P. Heymans, The ALBERT II specification animator, Technical Report, CREWS report 97-13, University of Namur (1997); <http://Sunsite.Informatik.RWTH-Aachen.DE/CREWS/>.
- [46] P. Heymans and E. Dubois, Scenario-based techniques for supporting the elaboration and the validation of formal requirements, *Requirements Engineering* 3 (1998) 202–218.
- [47] High-level Petri nets – Concepts, definitions and graphical notation, Final draft of international standard 15909, Version 4.7.1, ISO/IEC (28 October 2000).
- [48] J. Hodges and J. Visser, Accelerating wireless intelligent network standards through formal techniques, in: *IEEE 1999 Vehicular Technology Conf. (VTC'99)*, Houston, TX, USA, 1999; <http://www.UseCaseMaps.org/pub/vtc99.pdf>.
- [49] G.J. Holzmann, D. Peled and M. Redberg, Design tools for requirements engineering, *Bell Labs Technical Journal* 2(1) (1997) 86–95; http://www.lucent.com/minds/techjournal/winter_97/pdf/paper07.pdf; <http://cm.bell-labs.com/cm/cs/what/ubet/>.
- [50] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima and C. Chen, Formal approach to scenario analysis, *IEEE Software* (1994) 33–40.
- [51] R. Hurlbut, A survey of approaches for describing and formalizing use cases, Technical Report 97-03, Department of Computer Science, Illinois Institute of Technology, USA (1997); <http://www.iit.edu/~rhurlbut/xpt-tr-97-03.html>.
- [52] R.R. Hurlbut, Managing domain architecture evolution through adaptive use case and business rule models, Ph.D. thesis, Illinois Institute of Technology, Chigago, USA (1998); <http://www.iit.edu/~rhurlbut/hurl98.pdf>.
- [53] Information processing systems, open systems interconnection, LOTOS – A formal description technique based on the temporal ordering of observational behaviour, IS 8807, ISO, Geneva (1989).
- [54] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, *Object-Oriented Software Engineering, A Use Case Driven Approach* (Addison-Wesley/ACM Press, 1993).
- [55] M. Jarke and R. Kurki-Suonio, eds., Special Issue on Scenario Management, *IEEE Transactions on Software Engineering* 24(12) (1998).
- [56] F. Khendek and D. Vincent, Enriching SDL specifications with MSCs, in: *2nd Workshop of the SDL Forum Society on SDL and MSC (SAM'2000)*, Grenoble, France, June 2000.
- [57] K. Kimbler and D. Søbirk, Use case driven analysis of feature interactions, in: *Feature Interactions in Telecommunications Systems*, eds. L.G. Bouma and H. Velthuisen, Amsterdam, The Netherlands, May 1994 (IOS Press, Amsterdam, 1994) pp. 167–177.
- [58] J. Klose and H. Wittke, An automata based interpretation of live sequence chart, in: *Proc. of the 7th Internat. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, 2001.
- [59] K. Koskimies and E. Mäkinen, Automatic synthesis of state machines from trace diagrams, *Software Practice and Experience* 24(7) (1994) 643–658.
- [60] K. Koskimies, T. Männistö, T. Systä and J. Tuomi, SCED: A tool for dynamic modelling of object systems, University of Tampere, Department of Computer Science, Report A-1996-4 (July 1996); <ftp://cs.uta.fi/pub/reports/A-1996-4.ps.z>.
- [61] J. Koskinen, E. Mäkinen and T. Systä, Minimally adequate synthesizer tolerates inaccurate information during behavioral modeling, in: *SCASE 2001*, Enschede, The Netherlands, February 2001.
- [62] I. Krüger, R. Grosu, P. Scholz and M. Broy, From MSCs to statecharts, in: *Distributed and Parallel Embedded Systems* (Kluwer Academic, Dordrecht, 1999); <http://www4.informatik.tu-muenchen.de/papers/KGSB99.html>.

- [63] J.C.S.P. Leite, G.D.S. Hadad, J.H. Doorn and G.N. Kaplan, A scenario construction process, *Requirements Engineering* 5 (2000) 38–61.
- [64] S. Leue, L. Mehrmann and M. Rezaei, Synthesizing ROOM models from message sequence chart specifications, Technical Report 98-06, ECE Department, University of Waterloo, Canada (April 1998); short paper version in: *13th IEEE Conf. on Automated Software Engineering*, Honolulu, Hawaii, October 1998; <http://sven.uwaterloo.ca:80/~sleue/publications.files/tr98-06.ps.gz>.
- [65] J.J. Li and J.R. Horgan, Applying formal description techniques to software architectural design, *Computer Communications* 23(12) (2000) 1169–1178.
- [66] L. Liu and E. Yu, From requirements to architectural design – Using goals and scenarios, in: *From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, May 2001.
- [67] J. Magee and J. Kramer, *Concurrency: State Models and Java Programs* (Wiley, New York, 1999); <http://www-dse.doc.ic.ac.uk/~su2/Synthesis/>.
- [68] N.A.M. Maiden, SAVRE: Scenarios for acquiring and validating requirements, *Journal of Automated Software Engineering* 5 (1998) 419–446.
- [69] E. Mäkinen and T. Systä, MAS – An interactive synthesizer to support behavioral modeling in UML, in: *23rd Internat. Conf. on Software Engineering (ICSE'01)*, Toronto, Canada, May 2001.
- [70] N. Mansurov and R.L. Probert, A scenario-based approach to evolution of telecommunications software, *IEEE Communications* (October 2001).
- [71] N. Mansurov and D. Zhukov, Automatic synthesis of SDL models in use case methodology, in: *SDL'99, Proc. of the 9th SDL Forum*, Montréal, Canada (Elsevier, Amsterdam, 1999).
- [72] Message Sequence Chart (MSC), ITU-T Recommendation Z. 120, Geneva (2001); http://www.itu.int/ITU-T/studygroups/com17/languages/Z.120_1199.pdf.
- [73] Method for the characterization of telecommunication services supported by an ISDN and network capabilities of ISDN, ITU-T Recommendation I.130, CCITT, Geneva (1988).
- [74] A. Miga, Application of use case maps to system design with tool support, M.Eng. thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada (1998); http://www.UseCaseMaps.org/pub/am_thesis.pdf.
- [75] A. Miga, D. Amyot, F. Bordeleau, C. Cameron and M. Woodside, Deriving message sequence charts from use case maps scenario specifications, in: *10th SDL Forum (SDL'01)*, Copenhagen, Denmark, 2001; <http://www.UseCaseMaps.org/pub/sdl01-miga.pdf>.
- [76] OSI CTMF Part 3: The tree and tabular combined notation, 2nd ed, IS 9646-3, ISO/IEC, Geneva (1997).
- [77] D. Petriu and M. Woodside, Software performance models from system scenarios in use case maps, in: *12th Internat. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, London, UK (April 2002); <http://www.UseCaseMaps.org/pub/tools02.pdf>.
- [78] C. Potts, K. Takahashi and A.I. Antón, Inquiry-based requirements analysis, *IEEE Software* (March 1994) 21–32.
- [79] R.L. Probert and J. Saleh, Synthesis of communications protocols: Survey and assessment, *IEEE Transactions on Computers* 40(4) (1991) 468–476.
- [80] B. Regnell, Requirements engineering with use cases – A basis for software development, Ph.D. thesis, Department of Communication Systems, Lund Institute of Technology, Sweden, 1999; <http://www.tts.lth.se/Personal/bjornr/thesis/>.
- [81] B. Regnell, K. Kimbler and A. Wesslén, Improving the use case driven approach to requirements engineering, in: *Proc. of the 2nd IEEE Internat. Symposium on Requirements Engineering*, York, UK, March 1995, pp. 40–47; <http://www.tts.lth.se/Personal/bjornr/Papers/tts-94-24.ps>.
- [82] W. Reisig and G. Rozenberg, eds., *Lectures in Petri Nets*, Lecture Notes in Computer Science, Vols. 1491, 1492 (Springer, New York, 1998).

- [83] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyte, A.G. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymas, A proposal for a scenario classification framework, *Requirements Engineering Journal* 3(1) (1998) 23–47.
- [84] C. Rolland, C. Souveyet and C. Ben Achour, Guiding goal modelling using scenarios, Special Issue on Scenario Management, *IEEE Transactions on Software Engineering* 24(12) (1998).
- [85] A. Salah, R. Dssouli and G. Lapalme, Compiling real-time scenarios into a timed automaton, in: *Proc. of FORTE/PSTV'01*, China, 2001.
- [86] K. Saleh, Synthesis of communications protocols: An annotated bibliography, *ACM SIGCOMM Computer Communications Review* 26(5) (1996) 40–59.
- [87] K. Saleh, Synthesis of protocol converters: An annotated bibliography, *Computer Standards and Interfaces* 19 (1998) 105–117.
- [88] I. Sales and R. Probert, From high-level behaviour to high-level design: Use case maps to specification and description language, in: *SBRC'2000, 18^o Simpósio Brasileiro de Redes de Computadores*, Belo Horizonte, Brazil, May 2000.
- [89] S. Schönberger, R.K. Keller and I. Khriiss, Algorithmic support for model transformation in object-oriented software development, *Concurrency and Computation: Practice and Experience*, *Object Systems Section* 13(5) (2001) 351–383 (Wiley, New York, 2001).
- [90] SDL combined with UML, ITU-T Recommendation Z.109, Geneva (2000).
- [91] B. Selic, G. Gullekson and P.T. Ward, *Real-Time Object-Oriented Modeling* (Wiley, New York, 1994).
- [92] S. Somé, Dérivation de spécifications à partir de scénarios d'interaction, Ph.D. thesis, Département d'IRO, Université de Montréal, Canada (1997).
- [93] S. Somé, R. Dssouli and J. Vaucher, Toward an automation of requirements engineering using scenarios, *Journal of Computing and Information* 2(1) (1996) 1110–1132.
- [94] Specification and Description Language (SDL), ITU-T Recommendation Z.100, Geneva (2000); http://www.itu.int/ITU-T/studygroups/com17/languages/Z.100_1199.pdf.
- [95] T. Systä, Incremental construction of dynamic models for object-oriented software systems, *Journal of Object-Oriented Programming* 13(5) (2000) 18–27.
- [96] T. Systä, R.K. Keller and K. Koskimies, Summary report of the OOPSLA 2000 workshop on scenario-based round-trip engineering, *ACM SIGSOFT Software Engineering Notes* 26(2) (2001) 24–28; <http://www.iro.umontreal.ca/~labgelo/Publications/Papers/oopsla-2000.pdf>.
- [97] The unified functional methodology for the characterization of services and network capabilities including alternative object-oriented techniques, ITU-T Recommendation Q.65, Geneva (2000).
- [98] K.J. Turner, Formalising the chisel feature notation, in: *6th Internat. Workshop on Feature Interactions in Telecommunications and Software Systems (FIW'00)*, Glasgow, Scotland, UK, May 2000 (IOS Press, Amsterdam, 2000) pp. 241–256; <ftp://ftp.cs.stir.ac.uk/pub/staff/kjt/research/pubs/form-chis.pdf>.
- [99] S. Uchitel and J. Kramer, A workbench for synthesizing behavior models from scenarios, in: *23rd IEEE Internat. Conf. on Software Engineering (ICSE'01)*, Toronto, Canada, May 2001.
- [100] S. Uchitel, J. Kramer and J. Magee, Implied scenario detection in the presence of behaviour constraints, *Electronic Notes in Theoretical Computer Science* 65(7) (2002).
- [101] UCM: Use Case Map Notation, ITU-T, URN Focus Group, Draft Recommendation Z.152, Geneva (2002).
- [102] UML resource page, OMG (2002); <http://www.omg.org/uml/>.
- [103] Unified modeling language specification, Version 1.5, OMG (March 2003); <http://www.omg.org>.
- [104] Use case maps Web page, UCM User Group (1999); <http://www.UseCaseMaps.org>.

- [105] User Requirements Notation (URN) – Language requirements and framework, ITU-T: Recommendation Z.150, Geneva, Switzerland (2003); <http://www.UseCaseMaps.org/urn/>.
- [106] A. van Lamsweerde and L. Willemet, Inferring declarative requirements specifications from operational scenarios, Special Issue on Scenario Management of IEEE Transactions on Software Engineering 24(12) (1998) 1089–1114.
- [107] K. Weidenhaupt, K. Pohl, M. Jarke and P. Haumer, Scenarios in system development: Current practice, IEEE Software (March/April 1998) 34–45.
- [108] J. Whittle and J. Schumann, Generating statechart designs from scenarios, in: *22th Internat. Conf. on Software Engineering (ICSE 2000)*, Limerick, Ireland (ACM, New York, 2000) pp. 314–323.
- [109] J. Whittle and J. Schumann, statechart synthesis from scenarios: An air traffic control case study, in: *Scenarios and State Machines: Models, Algorithms, and Tools, ICSE 2002 Workshop*, Orlando, USA, 2002.
- [110] C.M. Woodside, D. Menascé and H. Gomaa, eds., in: *Proc. of the 2nd Internat. Workshop on Software and Performance (WOSP'2000)*, Ottawa, Canada, September 2000.
- [111] G.M. Yee and C.M. Woodside, A transformational approach to process partitioning using timed Petri nets, in: *Proc. of Internat. Computer Symposium 90 (ICS90)*, Taiwan, December 1990, pp. 395–401.
- [112] P. Zave and M. Jackson, Four dark corners of requirements engineering, ACM Transactions on Software Engineering and Methodology 6(1) (1997) 1–30; <http://www.research.att.com/~pamela/4dc.ps>.