# The Many Faces of Operationalization in Goal-Oriented Requirements Engineering

**Fabiano Dalpiaz[1], Vítor E. Silva Souza[2] and John Mylopoulos[3]**

[1] Utrecht University, P.O Box 80125, 3508 TC, Utrecht, the Netherlands
[2] Federal University of Espírito Santo, Av. Fernando Ferrari, 514/CT-7, Goiabeiras, Vitória ES, Brazil
[3] University of Trento, Via Sommarive 5, I-38123, Povo, Trento TN, Italy

f.dalpiaz@uu.nl, vitorsouza@inf.ufes.br, jm@disi.unitn.it

**Abstract.** Goal models have been used in Requirements Engineering (RE) to elicit, model and analyse stakeholder requirements. In a goal model, stakeholder requirements are represented as root-level goals that are iteratively refined through AND/OR-refinements to eventually yield a specification consisting of functions the system-to-be needs to implement, as well non-functional requirements and domain assumptions. The association of a function to a goal is called *operationalization* in the sense that the function specifies how a goal can be made operational. We focus on the concept of operationalization and propose several extensions to account for operationalizations of non-functional and adaptation requirements, as well as behavioural specifications.

*Keywords*: Goal-oriented requirements engineering, goal model, operationalization.

## 1 Introduction

Goal orientation in Requirements Engineering (hereafter RE) is founded on the premise that requirements are goals that stakeholders want fulfilled by the system-to-be. Goal orientation was proposed about 20 years ago in (Dardenne et al., 1993) as an improvement over traditional RE techniques that focused on the identification of functions that the system-to-be needs to implement. A goal model explains why are these functions needed and how they contribute to the fulfillment of what the stakeholders want. Moreover, each goal model defines a problem space that includes alternative ways of fulfilling root-level goals. Research on goal-oriented requirements engineering has been conducted in many research groups around the world, for example (Anton and Potts, 1998), (Kaiya et al, 2002), Yu and Mylopoulos, 1998), (Kavakli, 2002). An early survey of research on the topic can be found in (van Lamsweerde, 2001).

These ideas extended the software engineering process upstream, so that it starts with stakeholder wants/needs, rather than the functions the system-to-be needs to perform. Thanks to its acknowledged advantages, goal orientation has captured centre stage in RE research, as the dominant technique for requirements elicitation, modeling and analysis.

A goal model is constructed by iteratively *refining* the goals elicited from stakeholders into simpler goals through AND/OR refinements with obvious semantics: If a goal G is AND/OR-refined into subgoals $G_1$, $G_2$, …, $G_n$, then fulfillment of all/at least one of $G_1$, $G_2$, …, $G_n$, leads to fulfillment of G. Eventually, the subgoals obtained through refinements are simple enough (*atomic* or *leaf-level*) that they can be fulfilled by a function (aka action/task) that an external agent or the system-to-be can perform. Such functions *operationalize* leaf-level goals. Operationalizations cross the boundary between problem space (requirements modeled as goals) and the solution space (functional specification).

The main objective of this position paper is to focus on the concept of operationalization and propose extensions that have been found useful in using goal models to capture non-functional requirements (rather than functional ones) (Mylopoulos et al., 1992), also requirements for adaptive systems (Souza et al., 2013a,b) and behavioural specifications of requirements (Dalpiaz et al., 2013).

Our study reviews the fundamental concepts that goal models are founded on (Section 2), then sketches the history of operationalization in Natural and Social Sciences and proposes an extension intended to support the operationalization of non-functional requirements (Section 3). Section 4 proposes another extension to deal with adaptation requirements that introduce the monitor-analyze-plan-execute functionality that characterizes adaptive/autonomic software systems. In Section 5 we note the fundamental distinction between functional and behavioural specifications and introduce a new form of operationalization that maps a goal into a set of behaviours. Finally, Section 6 concludes.

## 2 Goal Models

Figure 1 shows a simple example consisting of a goal model obtained through refinements and operationalizations for a meeting scheduling system with a single stakeholder goal **`ScheduleMtg`**. In the example, fulfillment of the root-level goal can come about by fulfilling three subgoals. In turn, each one of these is OR-refined into two alternatives.

The three OR-refinements define choice points in the design and are labelled **`cp1`**, **`cp2`**, **`cp3`** respectively. For example, the fulfillment of goal **`CollectTMtables`** might be done by assigning the responsibility to a person who manually collects them, or to the system-to-be. In

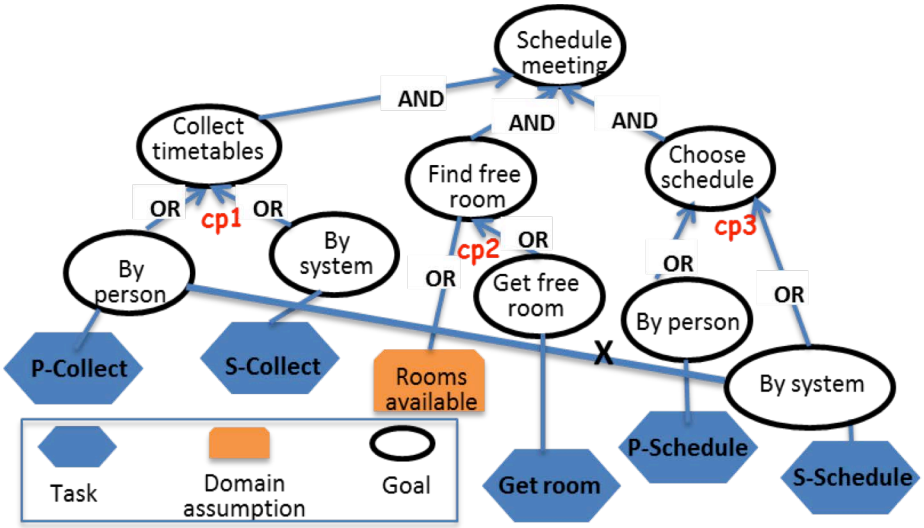


Figure 1: a simple goal model for the meeting scheduler

the former case, the responsible person has to execute function **`P-Collect`** with the system only providing database support. In the latter case, the system has to carry out **`S-Collect`**, which may involve automatically generated email messages for all anticipated participants, and sending reminders where appropriate.

The fulfillment of goal **`FindFreeRm`** also has two alternatives. The first one is a domain assumption: we can fulfill **`FindFreeRm`** by simply assuming that there will always be free rooms for a requested meeting. Alternatively, we can fulfill the goal **`GetRm`** that locates a free room. Domain assumptions simplify the design problem by assuming that some of the sub-problems will be solved by actors in the system's environment, so the designer doesn't have to worry about them.

Goal refinement (AND/OR) transforms a goal into one or more simpler ones. Eventually, these goals need to be operationalized ("made operational") either through the assignment of a function or the assignment of a domain assumption. In the former case, the designer is taking a proactive stance: "use this function to fulfill that goal". In the latter, the stance is opportunistic: "Something will happen to fulfill the goal, so we need not worry about it in our design". Operationalizations relate functions/ domain assumptions to the goals they operationalize. We call such operationalizations *functional* to distinguish them from what will be proposed in latter sections.

Another kind of link in Figure 1 is a conflict link (marked by "X"), which says that two goals/tasks/domain assumptions are in conflict to each other, so they can't be together part of a single solution. For instance, having a person collect timetables is in conflict with having the system choose a schedule for the meeting because not all collected timetables can be assumed to be in machine-readable form.

The goal model of Figure 1 suggests 6 possible solutions to the problem of scheduling meetings. The prefix of each element of the solution indicates whether it is a function (F) or a domain assumption (DA): **`{F:P-Collect, DA:RoomsAv, F:P-Schedule}`**, **`{F:P-Collect, F:GetRm, F:P-Schedule}`**, **`{F:S-Collect, DA:RoomsAv, F:P-Schedule}`**, **`{F:S-Collect, DA:RoomsAv, F:S-Schedule}`**, **`{F:S-Collect, F:GetRm, F:P-Schedule}`**, **`{F:S-Collect, F:GetRm, F:S-Schedule}`**. Such solutions are known as *functional specifications*.

A critical feature of goal models is that given a problem, e.g., fulfilling the goal **`ScheduleMtg`**, they define a space of alternative solutions, rather than a single solution. In this respect goal models (and feature models used to specify product families) are unique among models used in Software Engineering.

## 3 Qualitative Operationalization

The practice of operationalization (operationalism) was first proposed in Physics by Percy Williams Bridgman in 1927 (Bridgman, 1927). In short, operationalism calls for scientists to define their concepts, however abstract and intangible, in measurable terms. For example, "mass" might be operationalized in a gravity-oriented way as affinity to gravity measured by a weight scale. It can also be operationalized as resistance to force. Operationalism was subsequently adopted by the Life and Social Sciences where it provides measurable definitions for concepts such as "health" and "free and fair judiciary". For instance, "health" might be operationalized into a combined function of blood pressure, sugar level and number of drinks per day. "Free and fair judiciary" might be operationalized, on the other hand, in terms of the number of times there is government interference to judiciary functions, how often are members of the judiciary convicted of crime, etc.

For our purposes, operationalization of non-functional requirements amounts to adopting a precise measure by which an ill-defined non-functional requirement ("softgoal") can be measured as to the degree of its satisfaction. This is consistent with RE practice, where non-functional requirements are supposed to be "metricized" in terms of a metric. For example, a performance non-functional requirement might be metricized as "System shall process 1,000 transactions per second", and a usability requirement as "Users will be able to use the system after 3 hours of instruction".

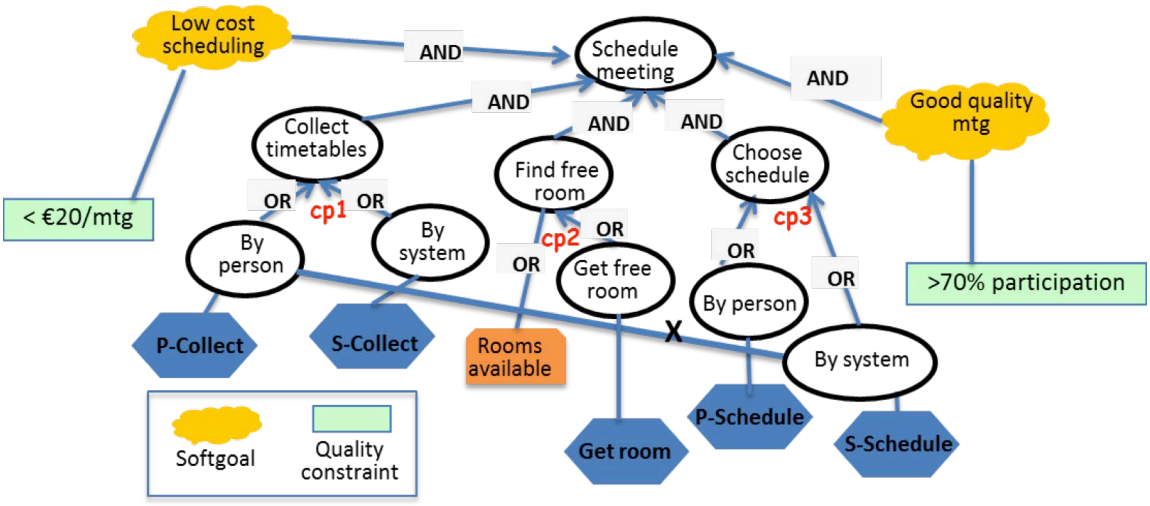


Figure 2: a goal model with softgoals and their operationalizations

Figure 2 shows an extended version of the meeting scheduler goal model with two softgoals representing non-functional requirements **`LowCostSch`** and **`GoodQuaMtg`**. These are operationalized into quality constraints ("metrics") 'Each meeting scheduling will cost ≤ €20' and 'Each meeting will have > 70% participation'. We call this type of operationalization *qualitative*, to distinguish it from its functional cousin.

## 4 Operationalizing Adaptation Requirements

The increasing complexity of software-intensive systems, combined with the uncertainty of the environments wherein they operate has made adaptive software systems a popular topic for researchers and practitioners alike (De Lemos et al., 2013).

To design adaptive systems, we need not only vanilla functional and non-functional requirements, but also *adaptation* requirements, such as "Meeting scheduling should not fail more than 2% during any one month period", or "If the collection of timetables fails because some participants did not respond, go ahead and schedule the meeting with the timetables you have". Such requirements are operationalized by a feedback loop that monitors the performance of the system and takes action when the requirement is not being met, i.e., meeting scheduling fails more than 2% of the time, or timetables haven't been collected from all participants.

Adaptation requirements come in two flavours. Awareness requirements (Souza et al., 2013a) impose constraints on the states (succeeded, failed, cancelled, etc.) of other requirements (i.e., goal model elements). For example, suppose that for the meeting scheduler three elements were found to be critical: **`ChooseSched`**, **`DA:RoomsAv`** and **`QC:70%Part`**. Further, each requirement has a different level of importance: **`ChooseSched`** should never fail, whereas we can tolerate one failure per week for **`DA:RoomsAv`** and would like a 75% success rate for **`QC:70%Part`**. These requirements for the monitoring component of the feedback loop are represented in Figure 3.

At runtime, the meeting scheduler should log changes of state of the instances of its goal model, e.g., "**`T:S-Sched`** has started", "**`T:S-Sched`** has succeeded/failed", etc. The feedback controller reads from this log, propagating the information up the model (following Boolean semantics of operationalization links), which may cause other elements to also change their state. These changes may eventually cause awareness requirements to fail, thereby triggering the system's adaptation mechanism.

Evolution requirements (Souza et al., 2013b) constitute another kind of adaptation requirement. Such requirements specify changes to other requirements when certain conditions apply. Evolution requirements are defined as Event-Condition-Action rules, taking relevant events from the monitoring component of the feedback loop and applying adaptation actions to the managed system, depending on certain conditions.

Figure 3 shows three evolution requirements for the meeting scheduler example. The first one uses failures of AR5 as the triggering event and is associated with two possible actions: (a) have the system retry the function that caused the failure; or (b) reconfigure the system. The condition for (a) is that it can only be applied once, whereas the condition for (b) is that (a) has been attempted but has failed to solve the problem.

The above example illustrates two kinds of adaptation action: evolution and reconfiguration. Evolution is used when stakeholders know exactly what the system should do in order to adapt. In this case, adaptation is defined as a series of modifications to the goal model, evolving it to represent a new problem space. Such modifications can take place at the instance level, which changes the system for a single user session (e.g., the retry case illustrated for **`ChooseSched`**), or at the requirement/class level, which changes the system from that point on (e.g., if requirement R fails more than we can tolerate, replace it with a less strict version R-).

Reconfiguration, on the other hand, can be applied when stakeholders do not have a specific solution to the problem, but would like the system itself to search in its solution space for alternative solutions (or *specifications*, as explained in Section 2). The three choice points of

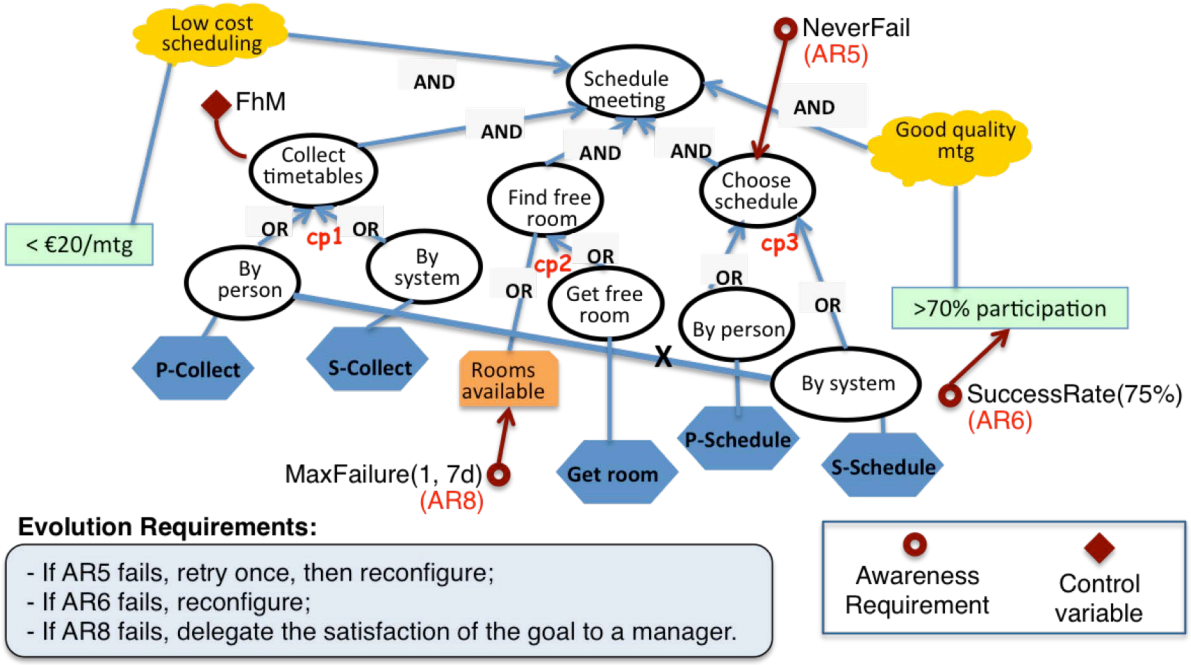


Figure 3: adaptation requirements operationalized by Awareness and Evolution Requirements

Figure 1 (**cp1**, **cp2**, **cp3**) allow the feedback controller to try different ways of satisfying system goals, therefore adapting to specific situations.

*Control variables* can also be identified as means for system reconfiguration. Figure 3 shows variable **FhM** connected to goal **CollectTMtables**. It prescribes From how Many participants one should collect timetables before moving on to schedule a meeting. It should be clear that changing the value of this variable could affect the satisfaction not only of goal **CollectTMtables**, but also of other requirements (e.g., the less timetables you collect, the higher the chance of poor quality meetings as participants find they can't attend a meeting they are supposed to). The relation between changes in these parameters (i.e., choice points and control variables) and the effect in satisfaction of requirements should also be elicited in order for the adaptation component of the feedback loop to be able to use this information properly.

Different reconfiguration algorithms have been proposed in the literature, each requiring different information to be included in a requirements model. (Dalpiaz et al., 2012) and (Souza et al., 2012) are two such examples from our own work.

## 5 Behavioural operationalization

Specifying a function through which a goal is operationalized is one way to move from the problem to the solution space, but there are also others. Behavioural specifications constitute one such alternative. *Behavioural* operationalizations define the possible behaviours of a system as the set of allowable sequences of executions of its functions (Dalpiaz et al., 2013).

Behavioural and functional operationalizations are complementary. Functional operationalization is applied to every leaf goal of a goal model and it defines the function through it can be fulfilled. Behavioural operationalization, on the other hand, applies to non-leaf goals and specifies in what order subgoals are to be fulfilled. For example, If goal G is AND-refined into subgoals $G_1$, $G_2$, a possible behavioural operationalization is '$G_1$ ; $G_2$', meaning that $G_1$ must be fulfilled first, followed by $G_2$. Alternatively, we may specify '$G_1$ | $G_2$', exactly one of the two subgoals needs to be fulfilled. More generally, we can use regular expressions of subgoals for behavioural operationalization. For instance, we may want to say that in order to schedule a meeting, we need to collect timetables one or more times until all timetables have been collected, then proceed with the scheduling

**CollectTMtables+ ; Schedule**

Here '+' stands for Kleene closure of regular expressions, while ';' indicates temporal ordering.

Behavioural expressions are actually more than regular expressions since we sometimes want to specify that two subgoals need to be fulfilled concurrently. This is indicated by the shuffle operator '#'. For example in Figure 4, meeting scheduling ($G_1$) is operationalized with '$G_2$; ($G_3$ # $G_4$)', indicating that **CollectTMtables** must be fulfilled first, followed by the interleaved fulfillment of $G_3$ and $G_4$. Likewise, the goal for collecting timetables ($G_2$) is annotated with '$(G_4 \mid G_5)^{\#}$', indicating that 1 or more versions of $G_4$ and $G_5$ can be fulfilled concurrently.

Functional operationalization tells us how to fulfill a leaf goal in terms of a function. On the other hand, behavioural operationalization tells us how to fulfill a non-leaf goal by using together solutions for its subgoals.

## 6 Discussion

Operationalization is about making something operational, either by providing a function that defines its operation, or by making it measurable. For goal models, these forms of operationalization cover the two basic types of requirements: functional and non-functional. In this paper we have examined other forms of operationalization that account for adaptation requirements and non-leaf goals.

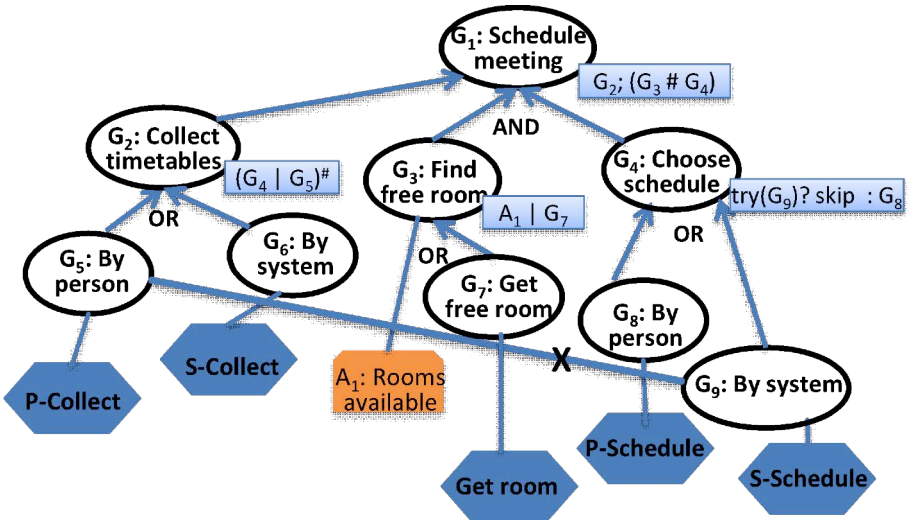


Figure 4: a runtime goal model that includes goal annotations

These extensions reflect different stances that a designer can take to the fulfillment of a requirement. A proactive stance amounts to providing the means to fulfillment through a function. An opportunistic stance assumes the problem away. A scientific stance delivers the means of measuring its degree of fulfillment. Finally, a reactive stance amounts to offering a mechanism (feedback loop) to cope with failures.

In conclusion, operationalization is a rich concept that has been in use in the Sciences for almost a century. It manifests itself in different ways for different classes of requirements. More importantly perhaps, it reflects multiple strategies to problem solving and design that go far beyond what has been explored and deployed in Requirements Engineering.

## Acknowledgements

This work was supported in part by ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution", http://www.lucretius.eu.

## References

Anton A.I. and Potts C. (1998): The Use of Goals to Surface Requirements for Evolving Systems. *Proc. of the 20th International Conference on Software Engineering*, Kyoto, Japan, 157-166, IEEE.

Bridgman P.W. (1927): *The Logic of Modern Physics*. New York, Macmillan.

Dalpiaz, F., Giorgini, P. and Mylopoulos, J. (2012): Adaptive socio-technical systems: a requirements-based approach. *Requirements Engineering* 18(1):1-24. Springer.

Dalpiaz, F., Borgida, A., Horkoff, J. and Mylopoulos, J. (2013): Runtime Goal Models. *Proc. of the 7th IEEE International Conference on Research Challenges in Information Science (RCIS)*, Paris, France, 1-11, IEEE.

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

De Lemos, R., Giese, H., Müller, H.A. and Shaw, M. eds. (2013): *Software Engineering for Self-Adaptive Systems II*. Berlin Heidelberg, Springer.

Kaiya H., Horai H., Saeki M. (2002): AGORA: Attributed Goal-Oriented Requirements Analysis Method. IEEE Requirements Engineering Conference, Essen, September 2002.

Kavakli E. (2002): Goal-Oriented Requirements Engineering: A Unifying Framework. *Requirements Engineering* 6(4):237-251. Springer.

Mylopoulos, J., Chung, L. and Nixon, B. (1992): Representing and using nonfunctional requirements: a process-oriented approach. *IEEE Transactions on Software Engineering* 18(6):483-497. IEEE.

Souza, V.E.S., Lapouchnian, A. and Mylopoulos, J. (2012): Requirements-Driven Qualitative Adaptation. In *On the Move to Meaningful Internet Systems: OTM 2012*. 342–361. R. Meersman et al., (eds). Springer.

Souza, V.E.S, Lapouchnian, A., Robinson, W.N. and Mylopoulos, J. (2013a): Awareness Requirements. In *Software Engineering for Self-Adaptive Systems II*. 133–161. De Lemos, R., Giese, H., Müller, H.A. and Shaw, M. (eds). Springer.

Souza, V.E.S., Lapouchnian, A., Angelopoulos, K. and Mylopoulos, J. (2013b): Requirements-driven software evolution. *Computer Science - Research and Development* 28(4):311-329. Springer.

van Lamsweerde A. (2001): Goal-Oriented Requirements Engineering: A Guided Tour. *Proc. of the 5th IEEE International Sysmposium on Requirements Engineering*, Toronto, Canada, 249-262, IEEE.

Yu E. and Mylopoulos J. (1998): Why Goal-Oriented Requirements Engineering. *Proc. of the 4th International Workshop on Requirements Engineering for Software Quality*, Pisa, Italy.