On the use of the Goal-Oriented Paradigm for System Design and Law Compliance Reasoning

Mirko Morandini[1], Luca Sabatucci[1], Alberto Siena[1], John Mylopoulos[2], Loris Penserini[1], Anna Perini[1], and Angelo Susi[1]

[1]Fondazione Bruno Kessler - IRST, Trento, Italy
{morandini,sabatucci,siena,perini,penserini,susi}@fbk.eu
[2]University of Trento, Italy
jm@disi.unitn.it

Abstract. The concept of goal may be used to model intentions of human actors, such as requirements analysts or designers, as well as the reasons for pro-active behaviour of software agents.
This short paper describes three ongoing research efforts on the application of the Goal-Oriented paradigm to system requirements analysis, system design and development of self-adaptive software agents.

Key words: Goal-Oriented paradigm, Requirements Engineering, Design Patterns, Software Agents

1 Introduction

The concept of *goal*, as a state of affairs that an actor (human, organization or system) wants to achieve, together with social aspects and other intentional concepts define the Goal-Oriented (GO) paradigm that has been largely studied and applied in Requirements Engineering (RE) since more than ten years [1, 2].

Evidences of the usefulness of the GO paradigm can be found in a variety of real world experiences that exploited available GO methodologies that support software system development activities ranging from requirements acquisition, analysis and understanding, to design and test cases derivation.

A key feature of the GO paradigm is that of allowing to model and reason about alternatives. These alternatives are usually represented in terms of OR-decompositions of goals or tasks, which lead to the definition of goal trees, whose alternative paths may be evaluated against possible situations of benefit, drawback or conflict.

This paper summarizes three ongoing research efforts at FBK-IRST in which the GO paradigm plays a central role at support of decision making for domain and system analysts, system designers and also for proactive artificial agents, in different contexts: (i) when analysts have to decide about the compliance of a set of the system's requirements with respect to law; (ii) when designers have to choose a suitable design pattern; (iii) when software agents of a self-adaptive system have to decide at run-time which one among their alternative sub-goals to achieve, while attempting to satisfy the main goals assigned to them by the designers.

2 An *i** Framework for Law Compliance: *Nòmos*

Laws and regulations address processes and associated information systems within organizations. Available methods and techniques for system design give little support to the requirements engineer when analysing the impact of those regulations during the definition of requirements for a new system, or when an existing organization has to restructure and re-engineer its operation in order to achieve compliance.

Goal-oriented requirements engineering rests on the idea of deriving the requirements for a software system from the analysis of the goals that the system-to-be will support once developed and deployed. However, when the stakeholders are addressed by laws, the system-to-be has to be aligned with the legal prescriptions, too, and goals per se do not provide information about such an alignment. This is the problem of law compliance of goals models. Finding a solution to this problem means finding the assignment of actors' responsibilities (goals) such that if every actor fulfils its goals, then law is respected. To address this problem we adopt a modelling approach which consists in starting from a model of legal prescriptions, and building the model of goals in an incremental way that maintains the alignment with the prescriptions.

The *i** modelling language focuses on intentional elements as the key to describe and understand a given organizational setting. Laws play a different role as they have (i) physical existence as natural language sentences in legal texts; and (ii) prescription objectives with regard to the organization. The contribution of the *Nòmos* framework consists in a conceptual binding between the two conceptions of intentions and regulations. The binding relies on the analysis of legal sentences, which ultimately allows to identify the juridical concept of **normative proposition** (NP), as the most atomic proposition able to carry a normative semantics, containing information concerning: (i) the subject(s) addressed by the NP itself; (ii) the legal modality; and (iii) the description of the object of such modality. The legal modality is one of the eight elementary rights, classified by Hohfeld [3] as: *Privilege*, which is the entitlement for a person to discretionally perform an action, regardless of the will of others and *Claim*, which is the entitlement for a person to have something done from another person, with their correlative rights *No-claim* and *Duty*; *Power*, which is the (legal) capability to produce changes in the legal system towards another subject, and *Immunity*, which is the right of being kept untouched from other performing an action, and their correlatives rights, *Liability* and *Disability.*

In order to support modelling of laws and compliance solutions, the *i** metamodel (in the variant proposed by the Tropos methodology [4]) has been extended with the *Nòmos* concepts, integrating the two set of concepts. Moreover, a systematic process has been defined to support the building of compliant requirements models, as described in detail in [5, 6], considering the following issues: *the binding of domain stakeholders with subjects addressed by law*, *the identification of legal alternatives*, *the identification of potential realisations of normative propositions*, *the identification of legal risks*, *the identification of proof artefacts*, *the constraining of delegation of goals to other actors.*

As future work, we are investigating the use of argumentation framework in order to support the acceptability of compliance solutions.

3 Design Pattern Representation with Motivations

Software patterns are reusable solutions to recurring design problems and — since their definition — are considered a mainstream of software reuse practice. They are typically documented with a textual description of the context where they can apply, the purpose for their reuse and forces to balance. For encouraging the understanding of design patterns and to ease their application during the design phase, many approaches have been proposed to provide the solution by using formal, semi-formal graphical notations or logic languages [7].

We propose to use i^* to represent not only the pattern solution, but also the whole reasoning process that led to its formation, including motivation, trade-offs and alternatives [8]. The main motivations of this approach are (i) to improve the communication encapsulated in a design pattern without changing the informative content and (ii) to provide some criteria for motivating pattern selection and reuse during the design process.

The proposed abstraction considers the design activity as the application area in which we apply the i^* framework, and the *Designer* as the main *Actor* of the design activity, whose job is to balance design forces coming from the system to be modelled. The designer's activities arise from needs, such as: (i) the achievement of *Design Goals* to solve specific design problem emerging during the modelling of the system, and (ii) the compliance with *Design Properties* (or soft goals) that specify qualities of the system. In this context a pattern is a collection of collaborating roles, intended as autonomous holder of design intentions that are delegated of some responsibilities from the designer, namely: (i) design goals/soft- goals to be achieved, (ii) design tasks for introducing a well-known solution, and (iii) system elements to introduce or to organize in the solution.

In this approach, the designer is supported with techniques for balancing pattern contextual forces and for customizing the pattern implementation to the specific application context. We exploit the Strategic Dependency model for representing the high-level responsibility organization of a design pattern, and the Strategic Rationale model for entering in detail in the solution structure.

The Strategic Dependency main role is always the designer who delegates design intentionality to pattern roles. This view allows for highlighting main intents and motivations of a pattern, and it is an instrument for quick selection of the pattern to reuse from a catalogue. On the other side the Strategic Rationale model allows for reasoning on design issues, considering consequences and balancing design alternatives, thus customizing the solution to meet forces coming from the context.

This approach opens new research directions we are working on: (i) to represent — and reason on — pattern composition and conflicts, and (ii) the use of an ontology for standardizing design intentional elements, that may help in automatic discovery of patterns to reuse.

4 Goal-Oriented Development of Self-Adaptive Systems

Self-adaptive Software (SAS) aims at dealing autonomously with unpredictable changes which occur in the dynamic environment it executes (at run-time), on the basis of its knowledge and of the objectives it has been designed for. We aim at defining a process and a tool-supported design framework to develop SAS with the necessary knowledge that enables adaptation of their behaviour at run-time. Belief-Desire-Intention (BDI) agents [9] were chosen as reference architecture and implementation platform.

The proposed framework and development process, called *Tropos4AS* (Tropos for Adaptive Systems) [10], exploits the basic Tropos early and late requirements phases, with an extensive use of variability modelling [11], and extends the design phase with environment modelling, extended goal modelling and failure modelling.

The *environment model* captures the non-intentional entities involved in, used and perceived by the SAS, which are necessary for interfacing the system with the surrounding world. For instance, the environment for a cleaner robot includes the floor, the dustbins and a battery charging station. These entities are represented through *artifacts* [12], non-intentional entities that provide functionalities usable by agents to sense and act in the environment.

In *extended goal modelling*, the goal model is linked with the environment. The process of goal achievement is related to the environment by defining the context where the goal is applicable, the conditions which lead to its adoption, its achievement and failure states. Different goal types further characterise the process of goal achievement, distinguishing among goals that the SAS has to achieve in a given situation (e.g. clean a wet floor), goals that the SAS has to achieve and maintain all along its life cycle (e.g. maintain a battery loaded), and goals that can be rather described as executing a given procedure (e.g. searching for dirt). Goal types were already present in the Formal Tropos [13] language, which was however developed with the aim of consistency verification via model-checking techniques. On the contrary, Tropos4AS focuses on the semantics of an agent's goal model that can be directly coded in a BDI agent programming language (e.g. Jadex, 2APL or Jack), following predefined mapping rules to link the goals in the design artefacts to the agent goals in the code, respecting the semantics of the goal model [14].

In *failure modelling*, Tropos4AS aids the designer in anticipating possible failures, giving a process to elicit errors possibly causing them and analysing the possibilities to fix them. Entities added to the Tropos4AS meta-model for supporting failure modelling are: *failures*, representing undesirable states known to the designer for the impossibility to achieve a goal, perceivable *errors* that may be the cause of these failures, and *recovery activities*, i.e. actions that the SAS may undertake to recover from errors, preventing failure.

The implementation is based on a tool-supported mapping of goal models to Jadex BDI agent code, maintaining the goal model with its semantics also at run-time. This goal model drive the agent's behaviour at a *knowledge level*, defining the relationship between requirements (goals) and the agents' capabilities, and to ensure traceability of run-time choices back to the design.

The main issue in our research agenda concerns the consolidation of the tools supporting the development and testing of self-adaptive software. Moreover, a validation of the framework on more realistic scenarios will be performed, focusing on the adaptive qualities of the system under development.

References

1. Mylopoulos, J., Chung, L., Yu, E.: From object-oriented to goal-oriented requirements analysis. Commun. ACM **42**(1) (1999) 31–37
2. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: RE, IEEE Computer Society (2001) 249
3. Hohfeld, W.N.: Fundamental Legal Conceptions as Applied in Judicial Reasoning. Yale Law Journal 23(1) (1913)
4. Susi, A., Perini, A., Mylopoulos, J., Giorgini, P.: The tropos metamodel and its use. Informatica (Slovenia) **29**(4) (2005) 401–408
5. Siena, A., Mylopoulos, J., Perini, A., Susi, A.: Designing law-compliant software requirements. In: 31st International Conference on Conceptual Modeling (ER'09), Gramado, Brasil (November 09 2009) 472–486
6. Siena, A., Mylopoulos, J., Perini, A., Susi, A.: Towards a framework for law-compliant software requirements. In: ICSE Companion. (2009) 251–254
7. Mikkonen, T.: Formalizing design patterns. In: Proceedings of ICSE '98, Washington, DC, USA, IEEE Computer Society (1998) 115–124
8. Sabatucci, L., Cossentino, M., Susi, A.: Introducing motivations in design pattern representation. In: Proc. of ICSR 11. LNCS, Springer (2009) 201–210
9. Rao, A.S., Georgeff, M.P.: Bdi agents: From theory to practice. In: ICMAS. (1995) 312–319
10. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. In: SEAMS '08: Workshop on Software engineering for adaptive and self-managing systems, New York, NY, USA, ACM (2008) 9–16
11. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: High variability design for software agents: Extending tropos. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **2**(4) (2007)
12. Omicini, A., Ricci, A., Viroli, M.: *Agens Faber*: Toward a theory of artefacts for MAS. Electr. Notes Theor. Comput. Sci. **150**(3) (2006) 21–36
13. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model checking early requirements specifications in Tropos. In: IEEE Int. Symposium on Requirements Engineering, Toronto (CA), IEEE Computer Society (August 2001) 174–181
14. Morandini, M., Penserini, L., Perini, A.: Operational Semantics of Goal Models in Adaptive Agents. In: 8th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'09), IFAAMAS (May 2009)