The Evolution of Tropos

John Mylopoulos, Jaelson Castro, and Manuel Kolp

Abstract The Tropos project was launched in the Fall of 1999 with main objective the development of a methodology for building agent-oriented software systems. The methodology was grounded on i* and was first presented in full at the CAiSE 2001 conference. This short article details some of the directions that were pursued in the project since that time.

1 Introduction

The Tropos project was launched in the Fall of 1999 with main objective the development of a methodology for building agent-oriented software systems. The methodology was grounded on i*, a requirements modeling language founded on intentional and social concepts, such as *actor, goal* and *social dependencies* among actors [18, 19]. Our CAiSE'01 paper was preceded by a couple of preliminary publications, most notably [12]. However, the CAiSE'01 paper constitutes the first comprehensive presentation of the Tropos methodology. Its publication was followed by two journal papers presenting further details on the methodology.

J. Mylopoulos (✉)
Department of Information Engineering and Computer Science, University of Trento, Via Sommarive, 14 – 38122 Povo – Italy
e-mail: jm@disi.unitn.it

J. Castro
Center of Informatics, Federal University of Pernambuco (UFPE), 50740-560 – Recife – PE, Brazil
e-mail: jbc@cin.ufpe.br

M. Kolp
School of Management, Catholic University of Louvain, Place des Doyens, B – 1348 Louvain-la-Neuve, Belgium
e-mail: manuel.kolp@uclouvain.be

J. Bubenko et al. (eds.), *Seminal Contributions to Information Systems Engineering*,
DOI 10.1007/978-3-642-36926-1_22, © Springer-Verlag Berlin Heidelberg 2013

Castro et al. [5] is an extended version of the CAiSE'01 paper included in the special issue published for CAiSE'01. Bresciani et al. [3] offers a complementary, more agent-oriented perspective on the modelling language and the accompanying methodology.

In what follows, we present in Sects. 2–4 some of the research questions that were pursued since 2001 at the University of Trento (Italy), the Catholic University of Louvain (Belgium) and the Federal University of Pernambuco (Brazil). The presentation is structured according to topic (Formal analysis, Architectures and Patterns, Methods and Techniques). We conclude with some of the research questions that the project has raised for the research community.

2 Formal Analysis

Much of the research that followed the CAiSE 2001 paper at the University of Trento focused on formal analysis techniques for Tropos models, Formal Tropos [7] extends Tropos by allowing annotations of i* models with Linear Temporal Logic (LTL) constraints. Formal Tropos models can then be translated into specifications for a model checker (in our case, nuSMV) to ensure that they satisfy desired formal properties. Along a parallel path, i* goal models were formalized so that one can check whether a goal model is satisfiable by using a SAT solver [15]. The PhD thesis of Volha Bryl uses an off-the-shelf planner to search among alternative ways of delegating an initial set of requirements among a group of actors and compares these alternatives using a number of local and global metrics for actor dependency networks [4].

Three more PhD theses extend the Tropos framework to address security concerns. Nicola Zannone's PhD thesis extends Tropos with concepts such as *ownership, permission* and *trust* forms of analysis that identify ownership/permission violations [8]. Yudis Asnar's PhD thesis, on the other hand, introduces risk-theoretic concepts to Tropos and looks at the problem of identifying suitable mitigation strategies for identified risks [1]. The thesis of Haralambos Mouratidis also extends Tropos to support secure software designs, but unlike Zannone, focuses on methodological extensions rather than ontological ones [11].

3 Architectures and Patterns

Along a different direction, Tropos has been extended to integrate organizational architectural styles [10] and social design patterns [9] for architectural and detailed design. Architectural styles are manageable abstractions that describe how system components interact and work together while design patterns describe a problem commonly found in software design and prescribe a flexible, reusable solution.

In Tropos, software architectures – be they multi-agent or component-based [13] – are considered social structures composed of autonomous and proactive entities that interact and cooperate with each other to achieve common or private goals. Since the fundamental concepts that drive Tropos are intentional and social, rather than implementation-oriented, theories that study social structures could provide inspiration and insights to define a catalogue of styles and patterns for designing software architectures with Tropos. For this, we turn for guidance to organizational theories, namely *Organization Theory* and *Strategic Alliances*.

Organization Theory describes the structure and design of an organization; *Strategic Alliances* model the strategic collaborations of independent organizational stakeholders who pursue a set of agreed upon business goals. Both disciplines aim to identify and study organizational styles. These are modeling abstractions that can be seen, felt, handled, and operated upon. They have a manifest form and lie in the objective domain of reality as part of the concrete world. A style is however not solely a set of execution behaviors. Rather, it exists in various forms at every stage of crystallization (e.g., specification), and at every level of granularity in the organization. The more manifest is its representation, the more the style emerges and becomes recognizable – whether at a high or low level of granularity.

Taking real-world social structures as metaphor, Tropos has then been extended to propose a set of generic architectural structures:

- At the architectural design level, organizational styles inspired from organization theory and strategic alliances will be used to design the overall system architecture. Styles from organization theory will describe the internal structure and design of the architecture, while styles from strategic alliances will model the cooperation of independent architectural organizational entities that pursue shared goals.
- At the detailed design level, social patterns drawn from research on cooperative and distributed architectures, will offer a more microscopic view of the social architecture description. They will define the software entities and the social dependencies that are necessary for the achievement of goals [9].

Mediation patterns constitute a particular category of social patterns featuring intermediate agents that help other agents reach agreements about an exchange of services. Mediation patterns include ones for monitor, broker, mediator, wrapper, embassy and matchmaker.

Although it is possible to reuse design solutions by using mediation patterns, current practices for instantiating these patterns in multi-agent system (MAS) development makes the application core highly coupled with the patterns' implementation, thereby reducing opportunities of reuse. To address this limitation, we proposed an agent-oriented design pattern description technique, called Agent Pattern Specifications (APS) [16], which takes into account the separation of pattern-related concerns in the MAS design level. A concern is some part of the problem that we want to treat as an integral conceptual unit. In addition, we used aspect-oriented programming to separate pattern-related concerns in the MAS implementation level. To do so, mapping guidelines were defined to guide the

implementation of patterns described according to APS by using an integration of JADE and AspectJ. This implementation was evaluated in terms of a suite of metrics for assessing well-known software engineering attributes, such as separation of concerns, coupling, cohesion and size. This assessment showed that aspect-oriented solutions for mediation patterns improved the separation of pattern-related concerns.

4 Methods and Techniques

Another line of research concerns the establishment of a relationship between requirements and architectural descriptions. The SIRA approach constitutes an initial proposal along this direction [2]. Both requirements and architectural designs are described in term of i* as actor dependency diagrams. One such diagram captures the social organization, while another captures a corresponding architectural organization. The organizational model, the main goals are identified by understanding a requirement model as the functionality requested for the system. The organization of the social system consists roles and interactions, as intended by the system and its environment. Additionally, goals and softgoals are used to select an organizational architectural style [10]. In the Assignment Model, roles are clustered into subgroups related to components, based on their similarity with the architectural components. The result is an architectural configuration, which is the allocation of sub-groups to architectural components.

The proliferation of iterative and incremental software development processes as de facto standards for SE practice, suggests a strong integration between requirements engineering and software architecture activities. Such integration can facilitate traceability and the propagation of changes between the models produced within these activities. Recognizing the close relation between architectural design description and requirements specification, we have advocated the use of model transformation approaches as an effective way to generate architectural models from requirements ones, where correlations between requirements and architectural models are accurately specified.

Hence, we have proposed STREAM, a systematic process for generating architectural models from requirements ones, based on horizontal and vertical transformations rules [6]. Horizontal transformations have source and target models at the same level of abstraction, while vertical transformations operate on models at different abstraction levels. In our case, the horizontal transformations are applied to the requirements models resulting in intermediary requirements models closer to architectural concerns. Vertical transformations, on the other hand, map these intermediary models into architectural models. Architectural design activities involve the selection and application of architectural patterns that best satisfy non-functional requirements. In STREAM, requirements models are described in i*, whereas architectural models are described using the Acme ADL.

Some quality attributes, such as adaptivity, are known to have an impact on the overall architecture of a system, so they need to be properly handled since

the beginning of the development process. Accordingly, we have proposed a new process called STREAM-A that includes six activities [14]. The first three are related to requirements engineering: (a) requirements refactoring; (b) context annotation and analysis; and (c) identification of sensors and monitors. The last three activities are architecture-related: (d) generate architectural model; (e) define architectural model; and (f) introduce a self-adaptation component.

Also, on the methodological process level, Tropos has been extended to offer iterative and incremental software project management [17]. *"I-Tropos development"* is an extension of the Tropos methodology that supports iterative and agile development.

The notions of *phase* and *discipline* are often presented as synonyms in the software engineering literature. Indeed, Tropos is described as composed of five phases (Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation). However, a discipline can be defined as a collection of activities that are all related to a major "area of concern", while phases here are not the traditional sequence of requirements analysis, design, coding, integration, and test. They are completely orthogonal to traditional phases. Each phase is concluded with a major milestone. In order to be compliant with the most generic terminology, traditional Tropos phases are then called disciplines in the I-Tropos process description since "they partition activities under a common theme". In the same way, phases are considered as groups of iterations that are workflows with a minor milestone. In I-Tropos, the Organizational Modeling and Requirements Engineering disciplines respectively correspond to Tropos' Early and Late Requirements disciplines. The Architectural and Detailed Design disciplines correspond to the same stages of the traditional Tropos process.

I-Tropos includes core disciplines, i.e., Organizational Modeling, Requirements Engineering, Architectural Design, Detailed Design, Implementation, Test and Deployment but also supports disciplines to handle Risk Management, Time Management, Quality Management and Software Process Management. For an iterative process, the need to support disciplines to manage the whole software project is of primary importance to precisely understand which project aspect to work on (and through which activity) at a specific time and with the best use of existing resources.

5 Conclusions

The focus of the Tropos project was agent-oriented software for good reasons. To the eyes of many in the Software Engineering and the Multi-Agent System communities, agent-orientation with its promise of autonomous, distributed, open computation seemed like a promising direction. In today's ever-more volatile world, our vision for the software systems of the future has been refined and placed into focus. We don't want just agent-oriented software systems, but rather socio-technical systems consisting of software, human and social actors that work together

to fulfill stakeholder requirements. The addition of human and social elements in the design has introduced new uncertainties that can only be addressed through adaptation mechanisms that our systems need to be endowed with. Perhaps more importantly, we don't aspire any more to design systems right from scratch. Rather, we expect to evolve systems continuously and so our methodologies have to be evolution-oriented. Like Darwin, we don't focus anymore on how software species came to be. Rather, we are interested in the ways they can evolve in order to survive.

References

1. Asnar, Y., Giorgini, P., Mylopoulos, J.: Goal-driven risk assessment in requirements engineering. Requirements Engineering Journal **16**(2), 101–116 (2011)
2. Bastos, L.R.D., de Castro, J.B.: Systematic integration between requirements and architecture. In: Springer (ed.) Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications, no. 3390 in LNCS, pp. 85–103 (2005)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3), 203–236 (2004)
4. Bryl, V., Giorgini, P., Mylopoulos, J.: Designing socio-technical systems: From stakeholder goals to social networks. Requirements Engineering Journal **14**(1), 47–70 (2009)
5. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the tropos project. Information Systems **27**(6), 365–389 (2002)
6. Castro, J., Lucena, M., Silva, C.T.L.L., Alencar, F.M.R., Santos, E., Pimentel, J.: Changing attitudes towards the generation of architectural models. Journal of Systems and Software **85**(3), 463–479 (2012)
7. Fuxman, A., Liu, L., Mylopoulos, J., Roveri, M., Traverso, P.: Specifying and analyzing early requirements in tropos. Requirements Engineering Journal **9**(2), 132–150 (2004)
8. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling security requirements through ownership, permission and delegation. In: Proceedings of the IEEE International conference on Requirements Engineering (RE'05), pp. 167–176 (2005)
9. Kolp, M., Do, T.T., Faulkner, S.: Social-centric development of multi-agent architectures. Journal of Organizational Computing and E-Commerce **18**(2), 150–175 (2008)
10. Kolp, M., Giorgini, P., Mylopoulos, J.: Multi-agent architectures as organizational structures. Autonomous Agents and Multi-Agent Systems **13**(1), 3–25 (2006)
11. Mouratidis, H., Giorgini, P., Manson, G.: Integrating security and systems engineering: Towards the modelling of secure information systems. In: 15th International Conference on Advanced Information Systems Engineering (CAiSE'03), Klagenfurt, vol. 2681, pp. 63–78. Springer-Verlag (2003)
12. Mylopoulos, J., Castro, J., Kolp, M.: Tropos: A framework for requirements-driven software development. In: Information Systems Engineering: State Of The Art And Research Themes, pp. 261–273. Springer-Verlag (2000)
13. Nguyen, T., Kolp, M., Penserini, L.: A development framework for component-based agent-oriented business services. International Journal of Agent Oriented Systems Engineering **3**(2/3), 328–367 (2009)
14. Pimentel, J.a., Lucena, M., Castro, J., Silva, C., Santos, E., Alencar, F.: Deriving software architectural models from requirements models for adaptive systems: the STREAM-A approach. Requirements Engineering Journal **17**(4), 259–281 (2012)

15. Sebastiani, R., Giorgini, P., Mylopoulos, J.: Simple and minimum-cost satisfiability for goal models. In: 16th International Conference on Advanced Information Systems Engineering (CAiSE '04), Riga, vol. 3084, pp. 20–35. Springer-Verlag (2004)
16. Silva, C., Castro, J., Araujo, J., Moreira, A., Tedesco, P., Mylopoulos, J.: Advanced separation of concerns in agent-oriented design patterns. International Journal of Agent-Oriented Software Engineering **3**(2–3), 306–327 (2009)
17. Wautelet, Y., Kolp, M., Poelmans, S.: Requirements-driven iterative project planning. In: S.B. Escalona Maria Jos Cordeiro Jos (ed.) Communications in Computer and Information Science, *Communications in Computer and Information Science*, vol. 303(6), pp. 121–135. Springer-Verlag (2012)
18. Yu, E.: Towards modeling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE '97, pp. 226–235. IEEE Computer Society, Washington, DC, USA (1997)
19. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J.: Social Modeling for Requirements Engineering. Cooperative Information Systems Series. Mit Press (2011)