Goal-Driven Agent-Oriented Software Processes

Carlos Cares
Llenguatges i Sistemes Informatics Dept. Technical University of Catalonia, Barcelona, Spain, and Ingeniería de Sistemas Dept., University of La Frontera, Temuco, Chile
ccares@lsi.upc.edu

Xavier Franch
Llenguatges i Sistemes Informatics Dept. Technical University of Catalonia, Barcelona, Spain
franch@lsi.upc.edu

Enric Mayol
Llenguatges i Sistemes Informatics Dept. Technical University of Catalonia, Barcelona, Spain
mayol@lsi.upc.edu

Enrique Alvarez
Ergonomic and Prevention Center (CEP) Technical University of Catalonia, Barcelona, Spain
enrique.alvarez@upc.edu

Abstract

The quality of software processes is acknowledged as a critical factor for delivering quality software systems. Any initiative for improving the quality of software processes requires their explicit representation and management. A current representational metaphor for systems is agent orientation, which has become one of the recently recognized engineering paradigms. In this article we argue for the convenience of representing the software process using an agent-oriented language to model it and a goal-driven procedure to design it. Particularly we propose using the i framework which is both an agent- and a goal-oriented modeling language. We review the possibilities of i* as a software process modeling language, and we also show how success factors can be made explicit in i* representations of the software processes. Finally, we illustrate the approach with an example based on the development of a set of ergonomic and safety software tools.*

1. Introduction

Software Process Improvement (SPI) has always been a permanent concern in software engineering research because following an adequate software process is acknowledged as a critical factor for delivering quality software systems. The design or re-design of software processes requires a representational model of them. Thus many ways to model them have been proposed: from highly detailed process-oriented modeling languages [1] to a simple sequence of activities. A main topic of research on software processes has focused on detecting and testing the critical success factors for SPI [2-5].

On the other hand, agent orientation is an emergent paradigm in software engineering. It has been recognized as a mainstream research area and there are many scholars that consider it "the" new paradigm in software engineering [6-9]. In the last few years, several agent-oriented software engineering methodologies have been proposed, such as: Prometheus, Gaia, Tropos, Roadmap and MaSE among others. A comparative analysis can be found in [10-12]. In these studies Tropos [13] is frequently considered to be in the group of the most relevant agent-oriented software methodologies. *i** [14] is the modeling language of Tropos and throughout its evolution it has included constructs for the classical concepts of both goal and agent orientation, namely: *goal*, *actor*, *agent*, *role*, *task*, *belief*, social dependency and commitment, among others.

In spite of the existence of these methodologies and the novelty of the paradigm, in [10] it is showed that all of these methodologies adjust their process models to classical software processes such as: waterfall, evolutionary, spiral and transformational. If we consider that a software methodology can be characterized by a modeling language and the software process [15], this observation leads us to say that the

Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06)
0-7695-2594-6/06 $20.00 © 2006 IEEE

new agent-oriented methodologies have made a significant contribution in their modeling languages but they have stayed static under the process perspective.

On the other hand, in [16] it is argued that the existing research suggestions for software process modeling have not been transferred into industrial use. As a possible solution, the improvement of software process representational languages is suggested.

In this paper we present theoretical arguments, complemented with an example, to support the idea that an agent-oriented representation of the software process is relevant and convenient for managing software development projects. Particularly, we propose the use of the *i** framework to represent the software process and, therefore, to use it as an abstraction tool which allows the organization of the software process. In section 2 we relate the main social concepts of the *i** modeling language with the suggestions formulated in [16] for the representational frameworks for the software process. Also in this section we outline a goal-driven procedure to obtain a software process design. In section 3 we review the main critical success factors of SPI and we show how these factors can be represented using *i** constructs. We remark here that an *i** representation is oriented to accomplish functional and quality goals, and therefore, *i** representations of some of the critical success factors of SPI give a way of both considering and of reaching them. Moreover, we show how the main scientific recommendations in software process modeling can be considered in a flexible software process design. In section 4 we illustrate, using a real example, how organizational and quality software concerns can be organized and represented as a coordinated set of organizational and software process goals. Finally we conclude by drawing together the different analyzed aspects and summarizing the arguments that support the convenience of using a goal-driven approach for an agent-oriented modeling of the software process

2. Software process modeling using *i**

The *i** framework [14] proposes the use of two models. The Strategic Dependency (SD) model allows the representation of organizational actors, their specializations (*role*, *position* and *agent*) and their relationships (*is-a*, *is-part-of*, *covers*, *plays* and *occupies*). The actors can have social dependencies between them that are characterized by a resource, task, goal and *softgoal*. A *softgoal* represents a goal, which can be partially satisfied, or a goal that requires additional agreement about how it is satisfied. They have usually been used for representing non-functional requirements and quality concerns. The second model is the Strategic Rationale (SR) model, which explodes the SD model. The SR represents the internal actors' *tasks*, *resources* and *goals* and their relationships such as *means-end*, contributions and decompositions. Moreover, many related approaches which create or modify *i** constructs for different objectives have been proposed, e.g. security concerns, stakeholder simplifications and types of commitments. These could be consistently added using the proposal from [17].

The basic idea of using the *i** framework as a software process modeling language was formulated in 1994 as an example of *i** constructs [18]. Throughout the last decade the research community has been formulating a new set of constructs and uses for *i**, however we do not know of any proposals that give suggestions or guides to the software process.

Table 1. Process elements and i* representation

Element	*i** representation
1. Activity	*Task*
2. Product	*Resource*
3. Role	*Role*
4. Human	*Agent* construct *(representing a human agent)*
5. Tool	*Resource,* which can be a means for achieving a particular *task*. Tool functionalities can also be represented by *tasks*.
6.Evolution Support	Process evolution can be supported by re-designing the process model and justifying it by using organizational goals, softgoals and beliefs and conceptually tracing the language modifications [1].
7. Project Organization model	*Actors'* model representing the structure of the organization (generally by using *actors* and the *is_part_of* actor relationships) for departments, sections and positions.
8. Work Context	The *actor* boundaries distinguish the internal representation from the contextual one.
9. User-view	The users and their *dependencies* on actors should constitute the user-view of the process model.
10 Cooperation model	Mainly by using *dependencies* between actors (*task*, *resource*, *goal* or *softgoal* dependencies)
11. Versioning and transaction	The *i** framework does not have specific mechanisms for versioning, however these elements seem to be adequate for tools as e.g. [19].
12. Quality and performance model	The *i** framework is especially suitable for quality concerns. However it requires additional constructs to represent deadlines and budgets.

In order to match the modeling language capabilities with the requirement for software process representation, we have studied the summary of the 12 requirements for software process languages stated in [1]. In table 1 we summarize the match with *i** suggested representations.

Additionally, in [16] another set of suggestions has been proposed, in order that software process representations allow: incomplete and informal specifications; easy adoption; practitioners to incrementally build their process model, making it more formal in advanced organizational stages, where enactment and simulation could take place. To show how *i** can accomplish these general features, we present some models showing summarized examples which accomplish the mentioned characteristics. In figure 1 we illustrate a small model which shows a general actor, the software team, which has the softgoal - *users' point of view effectively considered* – which should be fulfilled.

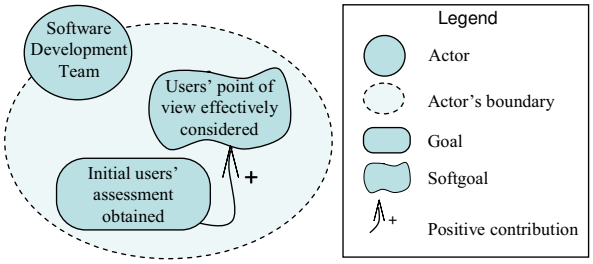

Figure 1. Partial and informal representation of a software process

In addition we also show the goal - *initial users' assessment obtained* - which contributes to accomplish the above *softgoal*. In this example we can see how there is not a specific role or person who is committed to this *goal*. Besides, the given *softgoal* is highly informal and the model does not represent complementary ways of contributing to the *softgoal*. In this sense we show that the software process model may be, initially, incomplete and informal.

We analyze the easy adoption feature from a cognitive perspective more than an organizational perspective. The organizational perspective is much more complex (e.g. the factors analyzed in [20, 21]). We analyze it from the most relevant success factors of SPI in the section 3. We think that the cognitive perspective can be analyzed using four factors: the design metaphor, the abstraction level, the notation and the formality of the proposal. First, the design metaphor of the *i** framework includes real life concepts such as: task, resource, actor, goal, dependency, and so on. Therefore these features imply that the *i** framework would be easier to adopt than others with no evident match between the reality and the modeling language. Second, the abstraction level of *i** allows both generic and specific representations and, moreover, both representations can coexist in the same diagram (e.g. the agent *Carlos* in figure 2). Third, the notation is mainly graphic which facilitates the interpretation of the diagrams. Although there are different specifications of the *i** framework, we have proposed a generic metamodel which defines the concepts and includes the different formalization approaches [17]. It can be considered a formal reference of the language in order to avoid potential ambiguities. These four reasons seem sufficient to argue that, from a cognitive perspective, *i** has relevant features that may make its adoption easier in a software development team.

Following the characteristics expressed in [16], the possibility to make software processes more formal (in the organizational sense) is also accomplished by the *i** framework. For example in figure 2, a specific organizational position into the developing team (*quality engineer*) is recognised, which covers a specific role (*usability tester*) and this role has been assigned the goal *initial users' assessment obtained*. In this example we have also represented the social (*task*) dependency from the generic actor user. Also we have introduced a means-end relationship, which identifies the means (in this case a task) which is useful to accomplish the identified goal.

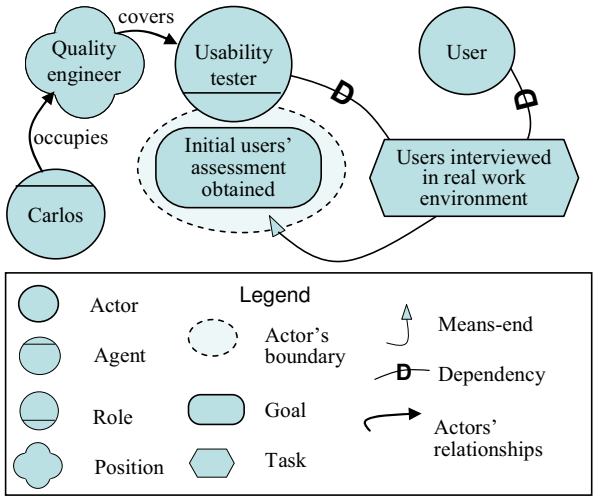

Figure 2. Organizational representation

With this second example we have shown that some classical organizational formalism, such as *positions*, *roles* and *tasks*, can be represented in a specific software process design and, moreover, the *i** representation can constitute the formalization of the organizational process itself.

Now we analyze the recommendation from [16] about the possibility of supporting enactment and simulation. For this we suggest that a first and initial organizational step is to apply the goal-verification process proposed in [22] which is a high level algorithm and does not require a detailed specification. However, if the organization reaches an advanced maturity level, this verification process can evolve to the temporal logic based analysis proposed in [23], which allows goal fulfillment simulation.

The final analyzed recommendation from [16] is about the possibility of building the software process incrementally. We have already stated that an *i** diagram could be informal and incomplete. However we think that the procedure for completing and detailing the diagram should be a goal-driven one and, this way, to incrementally obtain the software process representation. We propose the following stages to obtain the software process design: (1) represent the high-level organizational goals; (2) represent the high-level software process goals; (3) identify the needed roles to achieve the organizational goals and the new software process goals; (4) link organizational goals with software process goals identifying synergies (by contributions and/or goal decompositions); (5) identify the goal-dependencies between actors (identified roles and external actors such as stakeholders); (6) design the positions covering the roles with a balanced workload; (7) assign agents to identified positions; (8) identify additional high-level dependencies between agents (resource and task dependencies); (9) state internal goal and task decompositions considering the agent capabilities (software and related professionals); (10) review the model consistency, acceptance and social commitment.

This goal-driven procedure is strongly inspired by goal-oriented requirement engineering [24, 25] and, moreover, it accomplishes the meta-process phases: (mp-1) process elicitation and requirement specification; (mp-2) process analysis; and (mp-3) process design. These phases for arriving at the process design have been specified in [1]. Thus we have formulated a prescriptive method oriented specifically to designing the software process. Moreover, we have used the *i** constructs giving a strong orientation to *i** software process modeling. In this case our aim was to propose an additional element to complete the analyzed recommendations for a software process modeling framework.

Therefore we think that the revised conditions for a software process modeling language presented in [16] and in [1] are well accomplished by the *i** framework.

However, it is possible to observe from the above examples and the suggested procedure that the initial goal elicitation is not a clear and objective process. On the other hand there is an additional recommendation in [16] for including concepts and guidelines on how to accomplish technological, engineering, organizational and economic support. In order to find a set of guidelines for the *i** software process modeling we analyze the SPI success factors

3. Representing SPI's success factors

In this section we review some suggestions about SPI critical success factors (CSFs) in order to select the most relevant ones and show how they can be represented in *i** diagrams.

In table 2 we have summarized the CSFs and some identified barriers (transformed in a positive way as CSFs). Together with these CSFs we have added a column which specifies our proposal for representing the corresponding factor using *i** language elements. Given that we have reviewed CSFs from various sources and that there are similar factors with different names and closely related success factors, we have unified some of them.

In [26] 6 critical success factors for SPI have been identified. This research was conducted by literature research, interviews and questionnaires done in organizations in Sweden. The identified CSFs were: (1) process documentation; (2) change management (of the process documentation); (3) management commitment; (4) user involvement; (5) process synchronization (mainly on input-output issues); (6) baselining. Although we have numbered the factors in order of importance, it is important to remark that the authors have acknowledged that this relative importance is hard to generalize because other research shows different findings.

In [4] another CSFs research is presented. This report formulated questions about critical factors for SPI which were then answered by literature review. The analyzed studies covered more than 150 companies. In addition this study included its own empirical research on 20 Australian companies in order to support the proposal of a SPI implementation maturity model. Thus 18 CSFs were selected. From them we have selected the most frequently cited, namely: senior management commitment (3), training and mentoring, staff involvement, staff time and resources, experienced staff (7), creating process action teams (10), encouraging communication and collaboration (9), assignment of responsibility of SPI (10) and clear and relevant SPI goals (1). We believe

that creating process action teams and assigning SPI responsibility are both involved in managing the improvement process.

In the cited research [4] a set of barriers were also identified, the most common ones are: lack of resources (11), organizational politics, SPI getting in the way of real work, and staff turnover. These last three factors need some additional analysis before entering putting or creating the respective item in the table.

First, in organizational sciences, *organizational politics* has two main interpretations [27]. One is the influence that is exercised aside from the work setting (positive or negative). The second interpretation refers to behavior that is strategically designed to maximize self-interest, which could include work activities (e.g. designing an evaluation system oriented to self-promotion). However if we examine the questions addressed in [4], the issues around organizational politics refer to management and staff support for SPI, which could be maintained by adequate promotion and by the culture of the organization. In this sense we have divided this factor into: management and staff commitment (3), and change management (2), which should include goals like adequate change promotion.

Second, to address the SPI factor *gets in the way of real work*, we examine the concept *work*. The implicit meaning is that producing software is "real work" while improving the software process does not. Therefore explicit management and staff commitment (3) which should include SPI might be enough to achieve this success factor.

Third, although staff turnover could be caused by many reasons, we have included them in the factor adequate staff compensations (12).

In [28] 10 CSFs are analyzed 5 times, from 1998 to 2002, at Samsung Electronics Co., Korea. The factors were: management commitment and support (3), staff involvement (3), providing enhanced understanding (1), tailoring improvement initiatives (13), managing the improvement process (10), stabilizing the changed process (2), encouraging communication and collaboration (9), changing agents and opinion leaders (8), setting relevant and realistic objectives (10) and unfreezing the organization (we think that this is represented by factors 2, 3, 12 and 13). The economic factor appears in SPI topics as resource dependencies and management commitment, however this topic requires additional attention because it is one of the main factors to simulate in advanced software process representations and it is related with the budget planning mentioned in section 2.

Table 2. CSFs summary and *i representation**

CSFs of SPI		*i** representation proposal
1.	Process documentation	Presenting an *i** diagram of the software process with clear SPI goals
2.	Change management	Creating the goal *Management change including stabilization and promotion*
3.	Management and staff commitment	Creating explicit dependencies between management actors, mainly resource dependencies for improvement activities.
4.	User involvement	Creating the goal *User involvement achieved*
5.	Synchronization	Creating (information) resource dependencies and using task sequence constructs
6.	Baselining	Detailing the goals of each role
7.	Training and experienced staff	Creating training goals and explicit dependencies between human resource department and recruitment process and the technical and experience requirement
8.	Change agents and opinion leaders	Creating the goal *Considering opinion leaders as change promoters*. In [29] it is said that *promoting change agents* is an suitable choice too.
9.	Communication and Collaboration	Creating explicit information resource dependencies and cross communication goals.
10.	Managing the improvement process	Creating the goal *Software process improvement managed* and identifying the component which allows its fulfillment
11.	Resource availability	Representing resource dependencies on resource managers.
12.	Adequate staff compensation	Creating the goal *Personnel preserved by interesting compensations*.
13.	Tailoring improving initiatives	Clearly specifying the means (tasks and resources) to accomplish the suggested improvement goals.

We think that this proposal of SPI factors with the corresponding *i** representations, constitutes a well-founded guideline for how to accomplish technological, engineering and organizational support.

To sum up, our proposal includes seven high-level goal identifications, four high-level dependencies, one general assertion about process documentation and one characterization of the goal composition and means-end analysis. We observe that the amount of CSFs which include goals and goal dependencies gives additional support to the claim that the resulting agent representation is obtained by a goal-driven procedure.

4. UPCTools example

UPCTools is a set of ergonomic, hygienist and ambient safety tools. These software tools have been developed at the Ergonomics and Prevention Centre (CEP) at the Technical University of Catalonia. Since 2000, 35 tools have been developed which have used by more than 1500 registered users. The CEP is an educational and research unit; furthermore it has objectives other than producing software.

We have analyzed this case because, from a superficial view, the CEP followed an ill-formulated software process, without clear stages or the specification of intermediate work products. Despite this the professionals coordinate their work to produce software together. At the same time the software tools solve technical problems, and the big user community has a very good perception of the quality of these tools. We consider this as a particular case where well coordinated organizational and software process goals imply the development of high-quality software tools. We observe that, in fact, a detailed software process was never specified; however all the involved professionals clearly know their own goals and the dependencies with other members of the team. Moreover, there is a permanent effort made by the unit head to relate CEP members' professional development and research and educational goals with the particular software process responsibilities. We also observe how these goals and social dependencies constitute an intentional network designed to accomplish the main organizational goals. In this way we have built our example following a reverse engineering procedure, reviewing mainly task, goals and social dependencies.

In figure 3 we have represented the existing design using the *i** framework. We illustrate the model of actors and the relationships between their main organizational goals and the contributions of developing software services for the users' technical community.

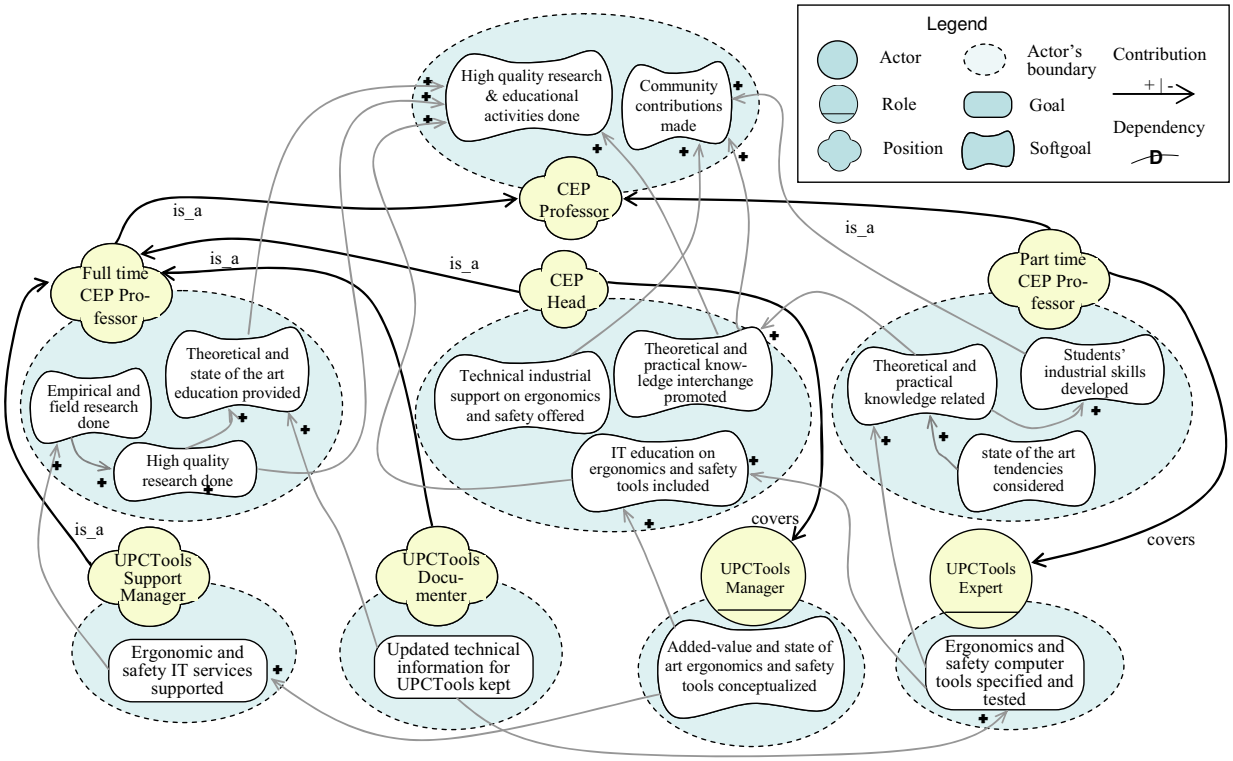

Figure 3. Software process intentional network obtained by reverse engineering at CEP

For example we highlight the positions of part-time professors, who are considered experts in the corresponding technical tools in his/her particular career, covering topics such as: ergonomic design, fire risk evaluation, vibration evaluation, among many others.

On the other hand a full time professor complements the work with a theoretical review, which is added into the documentation of the software tools. The tools are tested in practical educational activities which consist of generating safety and hygiene reports. At the same time these educational activities are oriented to stimulate the students' practical skills. Therefore the expert's work is supported by the theoretical revisions and practical tools of his/her corresponding teaching activities.

We have also represented the dependencies that we have added with the software professionals' roles and positions. We have observed how the dependencies tend to appear in pairs. For example the Software Developer (for a specific software tool) depends on the approval of the UPCTools Expert for technical issues, of the UPCTool Support Manager for user-interface consistency and of the Software Engineer for architectural design and consistency, among other dependencies. But the person responsible for technical issues is the UPCTools Expert, for architectural design is the Software Engineer and for user-interface consistency is the UPCTools Manager. We have observed these dependencies in the Strategic Dependency model which we have omitted here.

In this case we can see how a high-level goal organization implies a successful and long-term software production process. We also confirm our initial claim, that the agent-orientation and specifically the *i** framework, is adequate for representing software processes. Moreover, we think that our theoretical argument has initial empirical support, because we can see that a goal-driven software process, an agent-oriented representation, is suitable for tackling the problem of representing, designing and improving the software process.

5. Conclusions

We have proposed a goal-driven process to obtain agent-oriented representations of the software process. We have specifically analyzed the representational power of the *i** framework and we have related these capabilities to specific suggestions for software process modeling languages. We have presented four arguments to support our approach: (1) a comparative match among software process suggestions and *i** representational power, (2) a procedure inspired by goal-oriented requirement engineering aimed to guide the software process modeling, (3) an analysis of SPI factors to guide the initial goals and dependencies recognition and, finally, (4) a case study which illustrates both the desirability of a goal-driven approach and suitability of a agent-oriented representation of the software process.

This proposal is in line with the direction of our main research proposal, which is modelling knowledge reuse at the requirements stage. Therefore our future work will propose specific models to manage the requirements activities using the *i** framework. Also, we recognize that additional work should be done along related lines, such us empirical validation of the proposal, extra goal identification to improve the recommendation guidelines, including detailed economic issues in the representation of the software process. Also, it seems plausible and necessary to propose a framework which consistently mixes the abstraction level of the software process with the abstraction level of the domain analysis, especially when the *i** framework is used. Another future work suggestion is to deal with quality and certification concerns, because a goal-driven design procedure allows explicit goals from quality standards to be included.

There is some related work associated to OMG standards. In [30] and [31] the goal concept has been incorporated into the metamodel of the software process (SPEM). However, in both cases the interpretation is a low-level interpretation of this concept; it is at the operative level because it is associated with the *workproduct* class. Moreover the *goal* class is a subclass of the *constraint* class. In the case of [31] a *person-agent* is subclass of *agent*, however, following the metamodel, it implies that a *person-agent* is a subclass of *workproduct*.

Therefore, we conclude that our proposal is based on a more generic abstraction level, where software process actors (agents) can achieve management goals for software production and SPI. Therefore its convenience is not only the suitability of the representational language, but it is also that the main proposals for SPI can be represented at this abstraction level. In addition, we have complemented our approach with an example, which shows initial evidence that a well established intentional network constitutes a relevant topic at the manufacture of quality software products.

10. References

[1] R. Conradi and M.L. Jaccheri, "Process Modelling Languages", J-C. Derniame, B.A. Kaba, and D. Wastell, Eds, *LNCS,* Springer-Verlag Berlin, vol. 1500, 1999, pp. 27-52.

[2] S. Acuña et.al., "The Handbook of Software Engineering & Knowledge Engineering" in *The Software Process Modelling, Evaluation and Improvement*, vol. 1 (1): World Scientific Publishing Co., 2001, pp. 1-35.

[3] T. Dyba, "An Instrument for Measuring the Key Factors of Success in Software Process Improvement" *Empirical Software Engineering*, vol. 5, pp. 357-390, 2000.

[4] M. Niazi, D. Wilson, and D. Zowghi, "A maturity model for the implementation o software process improvement: an empirical study" *The Journal of Systems and Software*, 2003.

[5] D. Stelzer and W. Mellis, "Success Factors of Organizational Change in Software Process Improvement" *Software Process - Improvement and Practice*, vol. 4, 1998 pp. 227-250.

[6] P. Giorgini, "Agent-Oriented Software Engineering Report on the 4th AOSE Workshop" SIGMOD Record, vol. 32, 2003, pp. 117-118.

[7] N. R. Jennings, "On agent-based software engineering" *Artificial Intelligence*, vol. 117, 2000, pp. 277-296.

[8] J. Lind, "Issues in Agent-Oriented Software Engineering" *LNCS*, Springer-Verlag, Berlín, vol. 1957, 2001, pp. 45-58.

[9] X. J. Mao and E. Yu, "Organizational and social concepts in agent oriented software engineering," *LNCS,* Springer-Verlag Berlin, vol. 3382, 2005, pp. 1-15.

[10] L. Cernuzzi, M. Cossentino, and F. Zambonelli, "Process models for agent-based development", *Engineering Applications of Artif. Intell.*, vol. 18, 2005, pp. 205–222.

[11] K. H. Dam and M. Winikoff, "Comparing agent-oriented methodologies" *LNCS,* Springer-Verlag Berlin, vol. 3030, 2003, pp. 78-93.

[12] J. Sudeikat, et. al., "Evaluation of Agent–Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform", *LNCS*, Springer-Verlag Berlin, vol. 3382, 2005, pp. 126-141.

[13] J. Castro, M. Kolp, and J. Mylopoulos, "A Requirements-Driven Development Methodology" Advanced Information Systems Engineering: 13th Int. Conf. CAiSE, Interlaken, Switzerland, 2001, pp. 108-123.

[14] E. Yu, "Modelling Strategic Relationships for Process Reengineering," in Computer Science, vol. PhD. Toronto: University of Toronto, 1995.

[15] B. Bauer and J. P. Muller, "Using UML in the context of agent-oriented software engineering: State of the art", *LNCS*, Springer-Verlag, Berlin, vol. 2935, 2004, pp. 1-24.

[16] A. Fuggetta, "Software Process: A Roadmap", Proc. of the Int. Conf. on Software Engineering, ICSE 2000, Limerick Ireland, 2000.

[17] C. Ayala, et. al., "A Comparative Analysis of i*-Based Agent-Oriented Modeling Languages" Proc. of the 17th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Taipei, Taiwan, 2005, pp. 43-50.

[18] E. Yu and J. Mylopoulos, "Understanding "why" in software process modelling, analysis, and design", Proc. of the 16th Int. Conf. on Software Engineering, Sorrento, Italy, 1994, pp. 159-168.

[19] L. Jaccheri, J.-O. Larsen, and R. Conradi, "Software Process Modeling and Evolution in EPOS", *IEEE Trans. on Software Engineering*, vol. 19, 1992, pp. 1145-1156.

[20] R. Agarwal and J. Prasad, "A Field Study of the Adoption of Software Process Innovations by Information Systems Professionals", *IEEE Trans. on Engineering Management*, vol. 47, 2000, pp. 295-308.

[21] J. Becker, M. Rosemann, and C. v. Uthmann, "Guidelines of Business Process Modeling" in *Business Process Management: Models Techniques and Empirical Studies*, J. S. W. van der Aalst, A. Oberweis, Eds. Springer-Verlag, Berlín, 2000, pp. 30-49.

[22] F. Giunchiglia, J. Mylopoulos, and A. Perini, "The Tropos Software Development Methodology: Processes, Models and Diagrams", *LNCS*, Springer-Verlag, Berlín, vol. 2585, 2003, pp. 162-173.

[23] A. Fuxman, M. Pistore, J. Mylopoulos, and P. Traverso, "Model Checking Early Requirements Specifications in Tropos" Proc. of the 5th IEEE Int. Symp. on Requirements Engineering, Toronto, Canada, 2001, pp. 174-181.

[24] A. v. Lamsweerde, "Goal-Oriented Requirements Engineering: A guided Tour", Proc. of the 5th Int. Symp. on Requirements Engineering, 2001, pp. 249-261.

[25] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis", *Communications of the ACM*, vol. 42, 1999, pp. 31-37.

[26] P. Berander and C. Wohlin, "Identification of Key Factors in Software Process Management - A Case Study", Proc. of the Int. Symposium on Empirical Software Engineering (ISESE'03), Rome, Italy, 2003, pp. 316-325.

[27] R. Cropanzano, J. C. Howes, A. A. Grandey, and P. Toth, "The relationship of organizational politics and support to work behaviors, attitudes, and stress", *Journal of Organizational Behaviour*, vol. 18, 1997, pp. 159-180.

[28] S.-a. Lee and B. Choi, "Transition Management of Software Process Improvement" *LNCS*, Springer-Verlag, Berlín, vol. 2559, 2002, pp. 19-34.

[29] A. Rainer and T. Hall, "Key success factors for implementing software process improvement: a maturity-based analysis", *Journal of Systems and Software*, vol. 62 (2), 2002, pp. 71-84.

[30] "*Software Process Engineering Metamodel Specification, version 1.1*", Object Management Group, Inc., at http://www.omg.org/docs/formal/05-01-06.pdf, Last accessed March 2006, Feb. 2005.

[31] K. Cooper and L. Chung, "*Extending OMG Standards to Support Modeling Agents, Goals, and Components, Version 1.0*", CS Dept. U. Texas, Dallas, at http://www.utdallas.edu/~weiminma/public/folder/ Technical Report/UTDCS-41-04.pdf, Last accessed March 2006, 2004.