Enhancing Agent OPEN with concepts used in the Tropos methodology*

Brian Henderson-Sellers[1], Paolo Giorgini[2], and Paolo Bresciani[3]

[1] University of Technology, Sydney, NSW 2007 Australia,
brian@it.uts.edu.au,
[2] Department of Information and Communication Technology
University of Trento, Trento, Italy
paolo.giorgini@dit.unitn.it
[3] ITS-Irst, Povo (Trento), Italy
bresciani@itc.it

Abstract. Object technology has been supporting the development of information systems for many years but is now slowly evolving to encompass more recent ideas relating to the concept of "agent". Integrating agent concepts into existing OO methodologies has resulted in several agent-oriented methodologies, one of which is Agent OPEN. In this paper, we evaluate the existing Agent OPEN description against ideas formulated within Tropos, an agent-oriented software development methodology.

1 Introduction and Background

While object technology has been in widespread use for the development of information systems for many years, new ideas from the agent-oriented community are beginning to be addressed by extending existing OO methodologies to support the development of agent-based information systems (e.g., [26]). This is particularly evident in the discussions regarding whether agent orientation is a brand new paradigm requiring a non-OO mindset or whether it can be accommodated as an extension of existing OO ideas.

In this paper, we bring together one agent-oriented methodology which uses agent concepts throughout the process itself (Tropos [2]) and an object-oriented framework (the OPEN Process Framework or OPF [9]), specifically in its published extensions to support agents [6]. In Section 2, we present these existing extensions in the context of the OPF itself. In Section 3 we outline the Tropos methodology and then in Section 4 we evaluate whether the OPF in its extended form [6] is adequate to support the concepts and process elements described in Tropos and, where not, what further extensions are needed. We conclude in Section 5 with recommendations and outline directions for future work.

* This is Contribution Number 03/10 of the Centre for Object Technology Applications and Research (COTAR).

2 The OPEN Process Framework and its Existing Agent-oriented Enhancements

Integrating agent concepts into existing OO methodologies has resulted in several agent-oriented methodologies, for example, [7, 4, 26, 1]. One which we will discuss is the OPEN Process Framework, or OPF [11, 17, 9], which is a little different from most others in that it offers a metamodel-underpinned framework rather than (strictly) a methodology.

Method engineering (e.g., [3, 24]) is then used to construct project-specific or "situational" methods (a.k.a. methodologies). This is possible because of the provision of a repository of method fragments (e.g., [25]) or process components (e.g., [9]).

Initially, the repository of method fragments in OPEN was aimed at providing the ability to construct methodologies in the general area of information systems development. However, as new ideas emerged over the last few years, projects to extend the contents of the OPF repository have seen additions in areas such as component-based development [14], web-based development [13, 12] and organizational transition [16]. Initial extensions to agent-oriented development were formulated in [6, 15] and it is these extensions which we evaluate for completeness against the agent-oriented Tropos methodology — a comparison which is the focus of this paper.

2.1 The OPEN Process Framework

OPF consists of (i) a process metamodel or framework from which can be generated an organizationally-specific process (instance) created, using a method engineering approach [3], from (ii) a repository and (iii) a set of construction guidelines. The metamodel can be said to be at the M2 metalevel (Figure 1) with both the repository contents and the constructed process instance at the M1 level. The M0 level in Figure 1 represents the execution of the organization's (M1) process on a single project. Each (M1) process instance is created by choosing specific process components from the OPF Repository (Figure 1) and constructing (using the Construction Guidelines) a specific configuration — the "personalized OO development process". Then, using this method engineering approach, from this process metamodel we can generate an organizationally-specific process (instance).

The major elements in the OPF metamodel are Work Units (Activities, Tasks and Techniques), Work Products and Producers [9] — see Figure 2. These three components interact; for example producers perform work units, work units maintain work products and producers produce work products. In addition to these three metatypes, there are two auxiliary ones (Stages and Languages), which interact as shown in Figure 2.

Activity is at the highest level in the sense that a process consists of a number of Activities. Activities are largescale definitions of *what* must be done. They are not used for project management or enactment because they are at too high

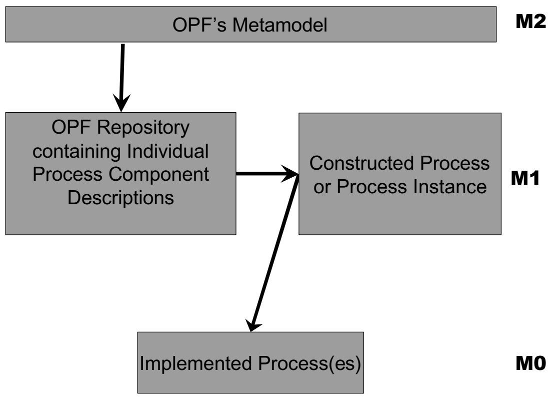
OPF's Metamodel
M2
OPF Repository containing Individual Process Component Descriptions
Constructed Process or Process Instance
M1
Implemented Process(es)
M0

Fig. 1. Three metalevels (M2, M1, M0) that provide a framework in which the relationship between the metamodel (M2), the repository and process instances (M1) and the implemented process (M0) can be seen to co-exist.

an abstraction level. Instead, OPEN offers the concept of Task (in agreement with the terminology of the Project Managers' Body of Knowledge [8]) which is defined as being the smallest unit of work that can be project managed. Both Activities and Tasks are kinds of Work Unit in the OPF metamodel (Figure 2).

Work Products are the outputs of the Activities. These work products may be graphically or textually described. Thus, we need a variety of languages to describe them. Typical examples here are English (natural language), UML (modelling language) and C# (implementation language). Since the metamodel itself is a "design model", it quite naturally is documented with one of the available modelling languages: here the Unified Modeling Language of the OMG [19] is the most commonly used (at least in OO developments).

While it is possible to analyze the metamodel directly, in this paper we address the issue of whether the contents of the current repository for the OPF is adequate for supporting agent-oriented developments. This repository contains *instances* generated from each of the metaclasses in the metamodel. For each metaclass there are potentially numerous instances. These are documented in various books and papers, as noted earlier. The ones specific to agents are listed in Table 1 (see next section).

2.2 The Current Agent OPEN

As a consequence of the modular nature of the OPEN approach to methodology, via the notion of a repository of process components together with the application of method or process engineering [23], it is relatively easy to add additional meta-elements and extremely easy to add additional examples of process components to the repository (as instances of pre-existing meta-elements). To

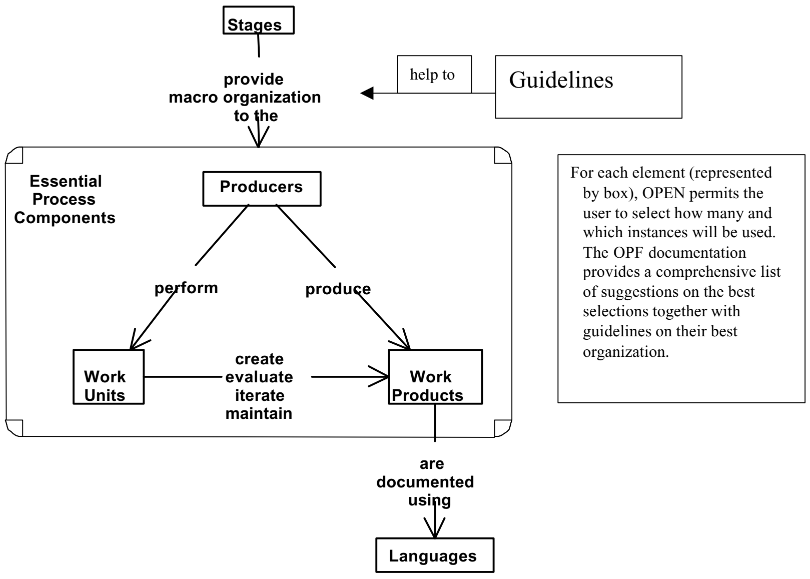
Stages
provide macro organization to the
help to
Guidelines
Essential Process Components
Producers
perform
produce
Work Units
create evaluate iterate maintain
Work Products
are documented using
Languages
For each element (represented by box), OPEN permits the user to select how many and which instances will be used. The OPF documentation provides a comprehensive list of suggestions on the best selections together with guidelines on their best organization.

Fig. 2. The five major metatypes in the OPF metamodel (after [9]).

extend this approach to support agent-oriented information systems, Debenham and Henderson-Sellers [6] analyzed the differences between agent-oriented and object-oriented approaches in order to be able to itemize and outline the necessary additions to the OPEN Process Framework's repository in the standard format provided in [17]. The focus of that work was primarily on instances of the meta-class WorkUnit useful for agent-oriented methodologies and processes. Table 1 lists the Tasks and Techniques so far added to the OPF repository (no new Activities were identified).

3 The Tropos Methodology

Tropos methodology [2, 10, 20] was designed to support agent-oriented systems development with a particular emphasis on the early requirements engineering phase. The stated aim was to use agent concepts in the description and definition of the methodology rather than using OO concepts in a minor extension to existing OO approaches. Tropos takes the BDI model [22, 18], formulated to describe the *internal* view of a single agent, and applies those concepts to the *external* view in terms of problem modelling as part of requirements engineering.

In Tropos [20, 5], AI derived mentalistic notions such as *actors*, *goals*, *softgoals*, *plans*, *resources*, and *intentional dependencies* are used in all the phases

Table 1 Tasks and Techniques already proposed [6, 15] for addition to the OPF repository in order to support the development of agent-oriented systems.

Tasks for AOIS	**Techniques likely to be useful**
Identify agents' roles	Environmental evaluation
Model the agent's environment	Environmental evaluation
Identify system organization	Environmental evaluation
Determine agent interaction protocol Determine delegation strategy	Contract nets Market mechanisms
Determine agent communication protocol	FIPA KIF compliant language
Determine conceptual architecture Determine agent reasoning	3-layer BDI model Deliberative reasoning: Plans Reactive reasoning: ECA Rules
Determine control architecture	Belief revision of agents Commitment management Activity scheduling Task selection by agents Control architecture
Determine system operation Gather performance knowledge	Learning strategies for agents
Determine security policy for agents	[topic of future research]
Undertake agent personalization	Environmental evaluation User model incorporation
Identify emergent behaviour	[topic of future research]

of software development, from the first phases of early analysis down to the actual implementation. It also includes descriptions of Work Products and several Techniques such as Means-End Analysis, useful in requirements engineering. A crucial role is given to the earlier analysis of requirements that precedes prescriptive requirements specification. In particular, aside from the understanding of *how* the intended system will fit into the organizational setting, and *what* the system requirements are, Tropos addresses also the analysis of the *why* the system requirements are as they are, by performing an in-depth justification with respect to the organizational goals.

4 Supporting Tropos Concepts in the OPEN Process Framework

In this section, we evaluate the existing Agent OPEN description (summarized in Section 2.2 above) against ideas formulated within the Tropos methodology, seeking any omissions or poor support of Tropos elements in the OPF. We then make recommendations for enhancements to the OPF in order that it can fully support all agent-oriented concepts formulated in Tropos.

Several new process components (method chunks) need to be added to the existing OPF repository. These are primarily Tasks and Techniques but there is also one new Activity: Early Requirements Engineering (in Tropos called the

Early Requirements Analysis phase) as well as some work products. All of these are outlined below in standard OPEN format and summarized for convenience in Table 2.

Table 2 Activity, Tasks, Techniques and Work Products proposed for inclusion in the OPF repository as a result of analyzing Tropos.

Activity	
Early requirements engineering	
Tasks	**Related Techniques**
Model actors	
Model capabilities for actors	Capabilities identification and analysis
Model dependencies for actors and goals	Contribution analysis
Model goals	Means–End Analysis Contribution Analysis AND/OR Decomposition
Model plans	Means–End Analysis Contribution Analysis AND/OR Decomposition
Work Products	
(Tropos) actor diagram (Tropos) capability diagram (Tropos) goal diagram (Tropos) plan diagram	

4.1 Activity

Early Requirements Engineering Early requirements engineering focusses on domain modeling. It consists of identifying and analyzing the relevant actors in organizations and their goals or intentions. These actors may correspond with the stakeholders but may also include other social elements (individuals, but also organizations, organizational units, teams, and so on) who do not directly share an interest in the project, but still need to be modelled in order to produce a sufficiently complete picture of the organizational domain. Each organization active element is modelled as a (social) actor that is dependent upon another (social) actor in order for them to achieve some stated goal. During Early Requirements Engineering, these goals are decomposed incrementally and finally the atomic goals can be used to support an objective analysis of alternatives.

The results of this analysis can be documented using a variety of Tropos diagrams. Goals, actors and dependencies can be depicted on an actor diagram and, in more detail, on a goal diagram. These results then form the basis for the “late requirements analysis” which in OPEN is called simply Requirements Engineering in which the system requirements are elicited in the context of the stakeholders’ goals identified in this activity of Early Requirements Engineering.

4.2 Tasks

Task: Model actors
Focus: People, other systems and roles involved
Typical supportive techniques: Business process modelling, Soft systems analysis
Explanation. While the concept of actors in OO systems already exist (and is supported in the original OPF), the Tropos methodology extends the OO notion of an actor beyond that of a single person/system/role interacting with a system to that of a more general entity that has strategic goals and intentionality within the system or organizational setting [2] including also, for example, whole organizations, organizational units and teams. Actors in Tropos can represent either agents (both human and artificial) or roles or positions (a set of roles, typically played by a single agent). This new Task thus considerably extends the existing concepts related to traditional OO actors. To model an actor, one must identify and analyze actors of both the environment and the system (or system-to-be). Tropos encourages the use of this Task in the early requirements phase for the modelling of domain stakeholders and their intentions as social actors. Actors can be depicted using (Tropos) actor diagrams (see below).

Task: Model capabilities for actors
Focus: Capability of each actor in the system
Typical supportive techniques: Capabilities identification and analysis
Explanation. The capability of an actor represents its ability to define, choose and execute a plan (for the fulfilment of a goal), given specific external environmental conditions and a specific event [2]. Capability modelling commences after the architecture has been designed, subsequent to an understanding of the system sub-actors and their interdependencies. Each system subactor must be provided with its own individual capabilities, perhaps with additional "social capabilities" for managing its dependencies with other actors/subactors. Previously modelled goals and plans generally now become an integral part of the capabilities. Capabilities can be depicted using (Tropos) capability diagrams and plan diagrams (see below).

Task: Model dependencies for actors and goals
Focus: How/if an actor depends on another for goal achievements
Typical supportive techniques: Contribution analysis, Delegation analysis
Explanation. In Tropos, a dependency may exist between two actors so that one actor depends in some way on the other in order to achieve its own goal, a goal that cannot otherwise be achieved or not as well or as easily without involving thsi second actor. Similarly, a dependency between two actors may exist for plan execution or resource availability [2]. The actors are named, respectively, the depender and the dependee while the dependency itself centres around the dependum. Dependencies can be depicted using (Tropos) actor diagrams and, in more detail, in goal diagrams (see below).

Task: Model goals
Focus: Actor's strategic interests
Typical supportive techniques: Means-end analysis, contribution analysis, AND/OR decomposition
Explanation. A goal represents an actor's strategic interests [2] — Tropos recommends both hard and soft goals. Modelling goals requires the analysis of those actor goals from the view point of the actor itself. The rationale for each goal relative to the stakeholder needs to be analyzed — typical Techniques are shown in Table 2. Goals may be decomposed into subgoals, either as alternatives or as concurrent goals. Plans may also be shown together with their decomposition, although details of plans are shown in a Plan Diagram (q.v.). Goals can be depicted using (Tropos) goal diagrams (see below).

Task: Model plans
Focus: Means to achieve goals
Typical supportive techniques: Means-end analysis, contribution analysis, AND/OR decomposition
Explanation. A plan represents a means by which a goal can be satisfied or, in the case of a soft goal, satisficed [2]. Plan modelling complements goal modelling and rests on reasoning techniques analogous to those used in goal modelling. Plans can be depicted using (Tropos) goal diagrams and plan diagrams (see below).

4.3 Techniques

Technique: Means–End Analysis
Focus: Identifying means to achieve goals
Typical tasks for which this is needed: Model goals, Model plans
Description. Means-end analysis aims at identifying plans, resources and goals as well as means to achieve the goals.
Usage. To perform means-end analysis, the following are performed iteratively:

– Describe the current state, the desired state (the goal) and the difference between the two
– Select a promising procedure for enabling this change of state by using this identified difference between present and desired states.
– Apply the selected procedure and update the current state.

If this successful finds an acceptable solution, then the iterations cease; otherwise they continue. If no acceptable solution is possible, then failure is announced.

Technique: Contribution Analysis
Focus: Goals contributing to other goals
Typical tasks for which this is needed: Model goals, Model plans
Description. Contribution analysis identifies goals that may contribute to the (partial) fulfilment of the final goal. It may be alternatively viewed as a kind

of means-end analysis in which the goal is identified as the means [2]. Contribution analysis applied to softgoals is often used to evaluate non-functional requirements.
Usage. Identify goals and soft-goals that can contribute either positively or negatively towards the achievement of the overall goal or soft-goal. Of course the focus is on identifying positive contributions, but the technique may also lead, as a side effect, to the identification of negative contributions. Annotate these appropriately (say with + or −). A + label indicates a positive, partial contribution to the fulfilment of the goal being analyzed. Contribution analysis is very effective for soft goals used for eliciting non-functional (quality) requirements.

Technique: AND/OR Decomposition
Focus: Goal decomposition
Typical tasks for which this is needed: Model goals, Model plans
Description. This is a technique to decompose a root goal into a finer goal structure.
Usage. Start with a high level goal and decompose into subgoals. These subgoals may either be alternatives (OR decomposition) or additive (AND decomposition).

Technique: Capabilities identification and analysis
Focus: Capabilities identification
Typical tasks for which this is needed: Model capabilities for actors
Description. For each goal introduced, we identify a set of capabilities that the responsible actor should have in order to fulfill the goal. When the achievement of the goal involves also other actors, the analysis is expended also to these actors and capabilities for the interaction/collaboration are identified and analyzed contextually(see [2] for more details).
Usage. Start with a goal associated to an actor and identify the capabilities needed locally. If the goal involves other actors the analysis is extended to these actors with respect to their contribution in the achievement of the goal.

4.4 Work Products

Work Product: (Tropos) Actor Diagram In Tropos, the actor diagram graphically depicts actors (as circles), their goals (as ellipses and clouds) attached to the relevant actor (Figure 3-a) together with a network of dependencies between the actors (Figure 3-b). In Figure 3-a, Citizen has two goals: the hard goal to “get information” and the soft goal to “ensure taxes well spent”. However, this soft goal is best delegated to the Information Bureau actor. To show this delegation, the delegated goal is shown explicitly as a dependum (cloud symbol) connected by two line segments to the two actors (Citizen and Information Bureau) (Figure 3-b).

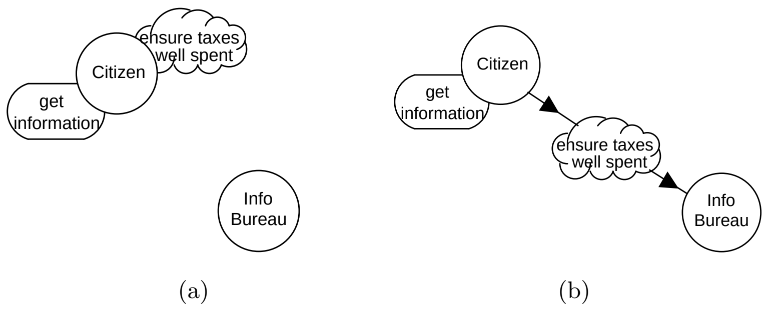

Fig. 3. (a) Example actor diagram showing goals attached to actors; (b) Example actor diagram showing an explicit dependee, depender and dependum

Work Product: (Tropos) Capability Diagram A capability diagram is drawn from the viewpoint of a specific agent. They are initiated by an event caused by an external event. Nodes in the diagram model plans (which can be expanded through the use of a Plan Diagram (q.v)) and transition arcs model events. Beliefs are modelled as objects [2]). Each node in the capability diagram may be expanded into a Plan Diagram (q.v.). Capability diagrams in Tropos use UML activity diagrams.

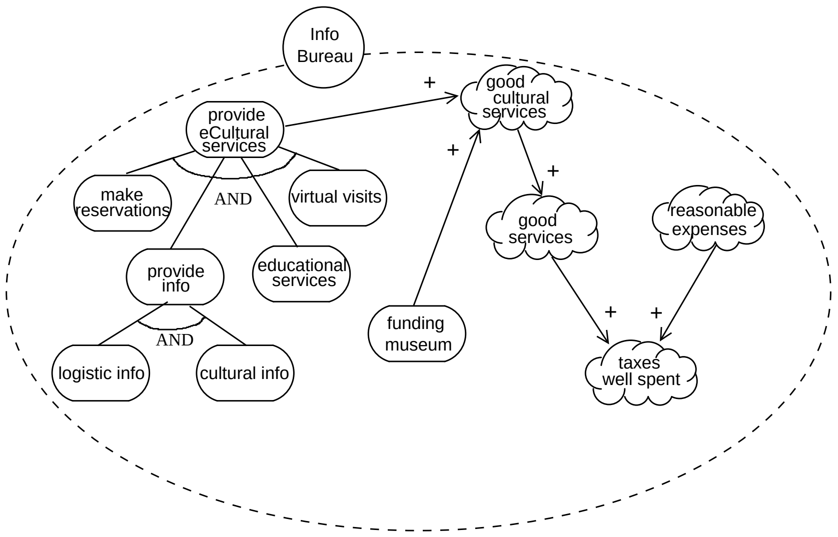

Fig. 4. Example goal diagram (after [2])

Work Product: (Tropos) Goal Diagram Figure 4 shows an example goal diagram in which the focus is that of how Information Bureau tries to achieve the delegated softgoal “taxes well spent”. Providing good services with reasonable expenses, Information Bureau can contribute to spend taxes well. Good services may include good cultural services, which in turn may include services available via web. So “provide eCultural services” can contribute positively in achieving the sofgoal “good cultural services”. Figure 4 shows also the partial AND decomposition of “provide eCultural services” goal.

Work Product: (Tropos) Plan Diagram A plan diagram depicts the internal structure of a plan, summarized as a single node on a Capability diagram (q.v.). Plan diagrams in Tropos use UML activity diagrams.

5 Conclusions and Future Work

Tropos extends methodological thinking into early requirements. It captures many aspects of agent-oriented requirements gathering not previously documented. In analyzing the extent to which other methodological frameworks, and in particular the OPEN Process Framework, supports these ideas, many deficiencies were identified. Here, we have itemized these gaps in OPEN's repository of process components and proposed additions to the repository specifically to address activities, tasks, techniques and work products found in Tropos but, until now, not available in the OPF repository.

We intend to progress these cross-fertilization between OPEN and Tropos, specifically taking advantage of the strengths of each: the early requirements engineering and agent focus of Tropos and the full lifecycle process of OPEN together with its metamodel-based underpinning that permits it to be used for situated method engineering.

References

1. Bernon, C., Gleizes, P.-P., Picard, G. and Glize, P., The ADELFE methodology for an intranet system design, Procs. Agent-Oriented Information Systems 2002 (eds. P. Giorgini, Y. Lespérance, G. Wagner and E. Yu), May 2002, Toronto, Canada.
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopolous, J. and Perini, A.: Tropos: an agent-oriented software development methodology. Journal of Autonomous Multi-Agent Systems (2003) in press
3. Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools. Inf. Software Technol. **38(4)** (1996) 275–280
4. Caire, G., Chainho, P., Evans, R., Garijo, F., Gomez Sanz, J., Kearney, P., Leal, F., Massonet, P., Pavon, J. and Stark, J., Agent-oriented analysis using MESSAGE/UML, Procs. Second Int. Workshop on Agent-Oriented Software Engineering (AOSE–2001), Montreal, Canada, May 2001, 101–107 (2001)
5. Castro J., Kolp M. and Mylopoulos J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*. Elsevier, Amsterdam, the Netherlands (2003) in press
6. Debenham, J. and Henderson-Sellers, B., Designing agent-based process systems — extending the OPEN Process Framework, Chapter VIII in Intelligent Agent Software Engineering (ed. V. Plekhanova), Idea Group Publishing (2003) 160–190
7. DeLoach, S.A., Multiagent systems engineering: a methodology and language for designing agent systems, Procs. Agent-Oriented Information Systems '99 (AOIS'99), Seattle, WA, USA, 1 May (1999) 1999
8. Duncan, W.R.: A Guide to the Project Management Body of Knowledge, Project Management Institute, PA, USA (1996) 176pp
9. Firesmith, D.G. and Henderson-Sellers, B.: The OPEN Process Framework. An Introduction, Addison-Wesley, Harlow, UK (2002) 330pp

10. Giorgini P., Perini A., Mylopoulos J., Giunchiglia F. and Bresciani P.: Agent-Oriented Software Development: A Case Study . Proceedings of the Thirteenth International Conference on Software Engineering and Knowledge Engineering (SEKE01), June 13-15 2001, Buenos Aires, Argentina (2001)
11. Graham, I., Henderson-Sellers, B. and Younessi, H.: The OPEN Process Specification, Addison-Wesley, Harlow, UK (1997) 314pp
12. Haire, B., Henderson-Sellers, B. and Lowe, D., Supporting web development in the OPEN process: additional tasks, Procs. 25th Annual International Computer Software and Applications Conference. COMPSAC 2001, IEEE Computer Society Press, Los Alamitos, CA, USA (2001) 383–389
13. Haire, B., Lowe, D. and Henderson-Sellers, B., Supporting web development in the OPEN process, Object-Oriented Information Systems (eds. Z. Bellahsène, D. Patel and C. Rolland), LNCS 2425, Springer–Verlag, 2002.
14. Henderson-Sellers, B., An OPEN process for component-based development, Chapter 18 in Component-Based Software Engineering: Putting the Pieces Together (eds. G.T. Heineman and W. Councill), Addison-Wesley, Reading, MA, USA, 2001.
15. Henderson-Sellers, B. and Debenham, J., 2003, Towards OPEN methodological support for agent oriented systems development, submitted for publication
16. Henderson-Sellers, B. and Serour, M., Creating a process for transitioning to object technology, Proceedings Seventh Asia–Pacific Software Engineering Conference. APSEC 2000, IEEE Computer Society Press, Los Alamitos, CA, USA, 2000.
17. Henderson-Sellers, B., Simons, A.J.H. and Younessi, H.: The OPEN Toolbox of Techniques, Addison-Wesley, UK (1998) 426pp + CD
18. Kinny, D., Georgeff, M. and Rao, A., A methodology and modelling techniques for systems of BDI agents, TR 58, Australian Artificial Intelligence Institute (1996)
19. OMG: OMG Unified Modeling Language Specification, Version 1.4, September 2001, OMG document formal/01-09-68 through 80 (13 documents) [Online]. Available http://www.omg.org (2001)
20. Perini A., Bresciani P., Giorgini P., Giunchiglia G. and Mylopoulos J.: A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In J. P. Müller, E. Andre, S. Sen, and C. Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, May 2001, Montreal, Canada, 2001.
21. Perini A., Bresciani P., Giorgini P., Giunchiglia F. and Mylopoulos J.: Towards an Agent Oriented approach to Software Engineering. In A. Omicini and M. Viroli, editors, *WOA 2001 – Dagli oggetti agli agenti: tendenze evolutive dei sistemi software*, 4–5 September 2001, Modena, Italy, Pitagora Editrice Bologna (2001)
22. Rao, A.S. and Georgeff, M.P., BDI agents: from theory to practice, Technical Note 56, Australian Artificial Intelligence Institute (1995)
23. Rupprecht, C., Fünffinger, M., Knublauch, H. and Rose, T., Capture and dissemiantion of experience about the construction of engineering processes, Procs. CAiSE 2000, LNCS 1789, Springer Verlag, Berlin, 294-308 (2000)
24. Ter Hofstede, A.H.M. and Verhoef, T.F., On the feasibility of situational method engineering, Information Systems, 22, 401-422 (1997)
25. van Slooten, K., Hodes, B., Characterizing IS development projects, in Proceedings of the IFIP TC8 Working Conference on Method Engineering: Principles of method construction and tool support (eds. S. Brinkkemper, K. Lyytinen, R. Welke) Chapman&Hall, Great Britain, 29–44 (1996)
26. Wooldridge, M., Jennings, N.R. and Kinny, D., The Gaia methodology for agent-oriented analysis and design, J. Autonomous Agents and Multi-Agent Systems, 3, 285–313 (2000)