AGENT-ORIENTED INFORMATION SYSTEMS DEVELOPMENT USING OPEN AND THE AGENT FACTORY

B. Henderson-Sellers, Q.-N. N. Tran, J. Debenham, and C. Gonzalez-Perez*

1. INTRODUCTION

Information systems development (ISD) requires the underpinning of a high quality methodology (which includes elements to describe both the process of development and the work products which are the consumables used and produced by the process). However, each ISD project is different and the best-fit methodology is also consequently different. This means that a one-size-fits-all methodology will only rarely give ideal results (Cockburn, 2000), when the tenets of the methodology designer coincidentally coincide with those of the particular project.

Rather than seeking an all-encompassing methodology, we advocate here the use of method engineering (Brinkkemper, 1996) or, preferably, situational method engineering or SME (Ter Hofstede and Verhoef, 1997). SME involves defining a repository of method fragments together with techniques for assembling these method fragments or method chunks (Rolland and Prakash, 1996) into site-specific methodologies specifically tuned to the situation of the project at hand (Brinkkemper, 1996) i.e. one that meets the requirements of a particular project. Thus, selection of method fragments is individualized and “tailored” to the specific requirements of the organization and project using construction guidelines supplied with the repository (Brinkkemper et al., 1998; Ralyté and Rolland, 2001). Many papers describing situational method engineering tend to focus on the process engineering element rather than the combination of process and product viz. the “methodology”. Since “process” is therefore a subset of “methodology”, when discussing only the “process” component of a methodology, the term *process engineering* is often substituted for the broader term “method engineering”.

For commercial adoption, the first choice is a widely used methodology framework with an existing extensive catalogue of method fragments. OPEN (Object-oriented Process,

* University of Technology, Sydney, P.O. Box 123, Broadway, NSW 2007, Australia.

Environment and Notation) is both described in an extensive set of books (e.g. Graham et al., 1997; Firesmith and Henderson-Sellers, 2002) and used in industrial applications as well as having (at least embryonic) tool support (Nguyen and Henderson-Sellers, 2003). While OPEN was originally developed for ISD in an object-oriented context, in recent years method fragments have begun to be developed for application to agent-oriented ISD (Debenham and Henderson-Sellers, 2003). In our current project, prior to a formalized extension of OPEN to fully support agent-oriented (AO) information systems development, we are analyzing each stand-alone AO methodology. For each such methodology, we seek to isolate method fragments identifiable in the methodology and then compare these with existing method fragments in OPEN/Agent OPEN. When there is no match, we suggest the development and incorporation in the repository of a method fragment to support the concept detailed in the AO methodology under analysis. Following successful analysis of Tropos (Henderson-Sellers et al., 2003, 2004b), MASE (Tran et al., 2004), Gaia (Henderson-Sellers et al., 2004a), Prometheus (Henderson-Sellers et al., 2004c) and Cassiopeia (Henderson-Sellers et al., 2004d), in this paper we look at the details of the Agent Factory (Collier et al., 2004).

In the following sections, we introduce situational method engineering (Section 2) as an effective approach for constructing an organizational method that may be tailored or customized for individual projects in the context of the OPEN process (Section 3). In Section 4 we outline briefly the Agent Factory approach and then analyze in detail in the following section (Section 5) in order to identify existing OPEN support for the Agent Factory set of concepts and to derive any necessary new method fragments.

2. BRIEF OVERVIEW OF SITUATIONAL METHOD ENGINEERING

Over recent years, there has been significant research into the field of method engineering and situational method engineering (Brinkkemper, 1996; Ter Hofstede and Verhoef, 1997; Rupprecht et al., 2000; Ralyté and Rolland, 2001). With SME, a method is constructed based on a methodological requirements statement made by the organization that requires methodological support for their software development. This requirements statement helps the method engineer to identify appropriate method fragments stored in the repository. Such an SME approach offers advantages to the use of a single “off-the-shelf” methodology since the method fragments in an SME repository not only support the methodology “as is” but also offer additional support that allow the methodology to be extended beyond its original intentions and scope. Alternatively, rather than extending an existing methodology, the more usual application of SME is to construct a methodology *ab initio*, specifically for the target situation.

In terms of SME research, the work by Brinkkemper et al. (1998) and Klooster et al. (1997) is also worth mentioning. These authors focus on the techniques for assembling available method fragments into situational methods/processes – an approach where a sound mathematical basis for process construction is described formally in a number of rules. The use of these rules allows the creation of a knowledge base, where knowledge can be stored and disseminated.

Ideally, the elements in an SME repository should be compliant with (in fact generated from) a set of concepts described by a metamodel (Henderson-Sellers, 2003). One

such example is the metamodel+repository-based OPEN process framework (Firesmith and Henderson-Sellers, 2002). The OPEN metamodel contains a number of conceptual entities modelled by object-oriented “classes”, typically described using the UML notational concepts. Those concepts most relevant to the process aspects of a methodology (as to be discussed here) are (i) Task, (ii) Technique and (iii) Work Product. Each of these metaclasses can be instantiated to create numerous instances of Task, Technique and Work Product respectively, all of which are stored in the OPEN repository. It is the elements of this repository that form the focus of our analysis here, in which we analyze existing process elements in the OPEN repository for their potential support for the Agent Factory approach to ISD.

3. BRIEF OVERVIEW OF OPEN

Unlike other OO ISD processes, OPEN (Graham et al., 1997; Firesmith and Henderson-Sellers, 2002) is defined as a process framework encompassing a metamodel. It is thus highly compatible with the ideas of method engineering and process construction as described above. From this framework, OPEN-compliant processes can be instantiated to be used in actual organization and software projects. In other words, the process engineer has to configure OPEN, creating an instance of OPEN that is suitable for use on a specific project. Instantiating OPEN is one of the more difficult and time-consuming jobs in adopting OPEN, since the process engineer has to understand the methodology, the organization, the environment and the software project itself in order to select the appropriate components in the OPEN repository to use on the project. Traditionally, this process is carried out using predefined organizational requirements and the experience and knowledge of the process engineer although tool support is likely in the near future (Saeki, 2003).

An important part of OPEN is the repository of process components, which can be used in different software projects. This repository provides an almost complete set of components and can be extended to support changes in technology.

The OPEN metamodel defines five main classes of process components as shown in Figure 1. From these (and their subclasses) are generated a significant number of instances, together with a number of guidelines (Firesmith and Henderson-Sellers, 2002) for helping organizations to adopt OPEN as a standard for their information system development projects. These guidelines offer advice on the selection of specific components based on the notion of deontic matrices. A deontic matrix is a two dimensional matrix of values that represent the possible relationship between each pair of process components in OPEN. For example, the possibility value for using the OPEN Task: Evaluate quality to help fulfil the Activity: Verification and Validation (V&V) might be assessed as being “Recommended” (one of the five prescribed values). Tool support for completing these matrices is currently under development (Nguyen and Henderson-Sellers, 2003).

In addition, the idea that business culture should be adapted to fit a specific methodology is not good business sense, despite its prevalence in many of the marketed methodologies to date. When using IT and its dependent parts, such as methodologies, it is critical that they fit the business and not the other way around.

Figure 1. The five major metaclasses of OPEN's metamodel (after Firesmith and Henderson-Sellers, 2002)
© Addison-Wesley.

4. BRIEF OVERVIEW OF THE AGENT FACTORY

Agent Factory (Collier et al., 2003, 2004) is a four-layer framework for designing, implementing, and deploying multi-agent systems. It contains (i) an agent-oriented software engineering methodology, (ii) a development environment, (iii) a FIPA-compliant runtime environment and (iv) an agent programming language (AF-APL); with a stated preference for the BDI agent architecture according to the analysis of (Luck et al., 2004). Here, we only evaluate the methodology components.

By employing UML and Agent UML, the Agent Factory methodology provides a visual, industry-recognized notation for its models - regarded by its authors as a major advantage over other approaches, such as Gaia (Wooldridge et al., 2000) and Tropos (Bresciani et al., 2004), which have non-standard (i.e. non-UML compliant) notations. These models are capable of promoting design reuse (via the central notion of *role*), and being directly implemented by automated code generation (Collier et al., 2004).

The Agent Factory describes three phases and an overall lifecycle approach. These, and their relationship to OPEN stages, are as follows:

Cycle: Agent Factory is iterative across the first three tasks of the development process (i.e. developing System Behaviour Model, Activity Model, and Interaction Model), but is sequential regarding the remaining steps.

Support from OPEN: A combination of "Iterative, incremental, parallel life cycle" (for iterative refinement) and "Waterfall life cycle" (for sequential development).

Life cycle: Agent Factory offers three phases, covering design, implementation and deployment. However the first three tasks of the Design phase (i.e. developing System Behaviour Model, Activity Model, and Interaction Model) are more appropriate to analysis rather than design activities. In other words, it appears that Agent Factory In fact more realistically can be said to cover four phases: analysis, design, implementation and deployment.

Support from OPEN: the three Agent Factory phases correspond to the Initiation, Construction, and Delivery phases of OPEN.

5. TASKS, TECHNIQUES AND WORK PRODUCTS IN THE AGENT FACTORY AND THEIR SUPPORT IN OPEN

In this section, we identify process component descriptions within the Agent Factory documentation, captured here as instances of elements in the OPEN metamodel. In particular, we seek Tasks, Techniques and Work Products. For each of these three elements, we analyze the Agent Factory descriptions and then recast them into the OPEN software engineering approach. This leads us to propose three new subtasks for addition to the OPEN repository as we extend this repository to encompass not only an object-oriented approach to software development but, increasingly, an agent-oriented approach. These new process components in the OPEN repository add to those already proposed to support agent-orientation in e.g. (Henderson-Sellers and Debenham, 2003; Henderson-Sellers et al., 2004b; Tran et al., 2004).

5.1. Tasks in the Agent Factory and Their Support in OPEN

For each Agent Factory task identified, we first describe it and then create a parallel OPEN method fragment.

5.1.1. Developing System Behaviour Model

Description: This task involves identifying the *key system behaviors* (i.e. *sets* of activities and/or interactions that occur during the operation of the system), and the *roles* that the agents will play while engaged in these behaviours.

Support from OPEN: The identification of system behaviours is supported by standard Requirements Engineering tasks in OPEN, particularly task “Use case modeling” (because Agent Factory captures system behaviors through use cases). Roles can be identified via the task “Model agent’s roles” in Agent OPEN (Debenham and Henderson-Sellers, 2003).

5.1.2. Developing Interaction Model

Description: This task investigates the system behaviours that involve interactions between two or more roles, and identifies all the potential “interaction scenarios” that may occur in each of these behaviours.

Support from OPEN: Agent OPEN offers tasks “Construct agent interaction protocol” and “Construct agent communication protocol” that can be extended to model interactions at a higher-level of abstraction (i.e. inter-role interactions).

5.1.3. Developing Activity Model

Description: This task investigates each system behaviour and identifies all potential “activity scenarios” that may occur within each behaviour. An activity scenario describes what activities need to be performed by the participant role(s) in order to realize a particular system behaviour.

Support from OPEN: The identification and modelling of activities within roles are addressed by Agent OPEN task “Model agent’s roles”, particularly by its sub-task “Model roles’ responsibilities” (Henderson-Sellers et al., 2004a).

5.1.4. Developing Protocol Model

Description: This task formalizes the interaction scenarios identified in the Interaction Model with interaction protocols. Each protocol elaborately defines the inter-role interactions in a particular system behaviour, encapsulating all possible variations.

Support from OPEN: This task can be supported by Agent OPEN’s tasks “Determine agent interaction protocol” and “Determine agent communication protocol”, although at the “role” level of abstraction rather than at the “agent” level.

5.1.5. Developing Agent Model

Description: This task identifies agent classes from roles, and models agent classes in terms of their activities, roles, and roles’ protocols.

Support from OPEN: Agent OPEN task “Construct the Agent Model” (Tran et al., 2004) directly supports this activity.

5.1.6. Defining Application-Specific Ontologies

Description: This task specifies the conceptualization of the target application domain via a (set of) ontologies. These ontologies are needed to form the beliefs of individual agents and the information exchanged between interacting agents.

Support from OPEN: The issue of ontology specification to be used in system operation is currently not addressed in OPEN. A new task is thus desirable

TASK NAME: Define ontologies

Focus: Domain conceptualization

Typical supportive techniques: Domain analysis

Explanation: A conceptualization of the target application domain needs to be specified. This conceptualization should contain all the concepts, entities, and relationships that exist in the domain, and which are relevant to the needs of agents in the system (e.g. communication and internal processing needs).

5.1.7. Building Agent Components

Description: This task generates agent components that are required by the final system, particularly perceptor and actuator units.

Support from OPEN: Agent OPEN recently introduced a task “Design agent internal structure” (Tran et al., 2004) that addresses the design of agent internal modules. We suggest adding two new sub-tasks to explicitly address the specification of perceptor and actuator modules. These are subtasks to the existing Task: Design agent internal structure.

SUBTASK NAME: Define perceptor module

Typical supportive techniques: Environmental evaluation

Explanation: Define for each agent its required sensing abilities, and mechanisms needed to convert raw data/percepts to beliefs. To promote reusability, sensing abilities and mechanisms can be packaged into perceptor components that are later associated to agents.

SUBTASK NAME: Define actuator module

Typical supportive techniques: Environmental evaluation

Explanation: Define for each agent the primitive actions that can be directly executed by the agent on the environment. To promote reusability, sets of primitive actions can be packaged into actuator components that are later bound to agents.

5.1.8. Building Platform Services

Description: This task identifies and constructs services that are required to be deployed on the agent platform, e.g. message transport, migration, persistence services.

Support from OPEN: The OPEN task “Create a system architecture” can be extended to include a new sub-task “Determine MAS infrastructure facilities”.

SUBTASK NAME: Determine MAS infrastructure facilities

Typical supportive techniques: Environmental evaluation

Explanation: Facilities required to support the operation of MAS as a whole and of agents should be identified, as well as how they are managed (e.g. by agents).

5.1.9. Implementing Agent Classes

Description: This task generates AF-APL code to implement agents.

Support from OPEN: The OPEN task “Code” can be extended to cover the coding of agent classes

5.1.10. Testing

Description: This task performs “protocol tests” on agent interaction protocols, and “behaviour tests” on agent behaviours, in order to evaluate their correctness.

Support from OPEN: A range of OPEN Testing tasks, including “Design test suite”, “Execute tests”, and “Report on test results”, can be extended to cater for agent-oriented testing.

5.1.11. Deployment

Description: This task involves configuring the agent development platform and deploying the application.

Support from OPEN: OPEN’s set of Deployment tasks, although initially intended to support the deployment of OO applications, can be equally applicable to AO application deployment.

5.2. Agent Factory Techniques and Their Existing Support in OPEN

For each Agent Factory technique identified, we first describe it and then create a parallel OPEN method fragment.

5.2.1. For Developing System Behaviour Model

Description: Regarding the identification of system behaviours, the developer should investigate both activity-oriented behaviours (i.e. those associated with a single role), and interaction-oriented behaviours (i.e. those associated with two or more roles). Regarding the identification of roles, Agent Factory offers no techniques.

Support from OPEN: Various standard OPEN techniques can be useful for system behaviours identification, including “Scenario development”, “Activity grid construction”, and “Service identification”. Role identification can be assisted by the conventional OPEN technique “Role modeling” (although this technique is still weak in guidance for role elicitation), and the newly-added technique “Environmental evaluation” in Agent OPEN (Henderson-Sellers and Debenham, 2003).

5.2.2. For Developing Interaction Model

Description: For each interaction-oriented system behaviour in the System Behaviour Model, the developer should define a number of “interaction scenarios”, each specifying a potential set of interactions that may incur within the behaviour. Each scenario should describe the types of messages sent among roles, and the order in which they are sent. There typically exist one “standard” scenario and multiple alternate scenarios for each interaction-oriented behaviour.

Support from OPEN: Various conventional OO techniques can be useful for the identification and specification of interaction scenarios, including “Scenario development”, “Collaboration analysis”, and “Interaction modeling”.

5.2.3. For Developing Activity Model

Description: The developer should specify at least one “activity scenario” for each activity-oriented system behaviour, and zero or more “activity scenarios” for each interaction-oriented behaviour. In each scenario, the activities to be performed by each participant role should be specified. Multiple activity scenarios exist when there are different ways to fulfil a behavior.

Support from OPEN: The identification and specification of activity scenarios can be supported by conventional OPEN techniques “Scenario development”, “Responsibility identification”, and “State modeling”.

5.2.4. For Developing Protocol Model

Description: Each interaction-oriented system behaviour typically requires one protocol to be specified. If multiple interaction scenarios have been identified for the behaviour in the Interaction Model, they can be integrated into a single protocol. The developer may make use of existing protocol templates, and/or formulate new templates by identifying and extracting common interactions within the defined protocols.

Support from OPEN: Conventional OPEN technique “Interaction modeling” and Agent OPEN techniques “Contract net”, “Market mechanisms”, and “FIPA-KIF compliant language” can be applied to specify the protocols and exchanged messages.

5.2.5. For Developing Agent Model

Description: Agent classes can be derived from roles via a many-to-many correspondence, i.e. each role can be mapped onto many agent classes, while each agent class can be mapped to multiple roles. Agents are associated with activities, which are determined by examining the potential “activity scenarios” for each system behaviour in the Activity Model, and selecting a scenario the agent should employ when realising a particular behavior.

Support from OPEN: Regarding the identification of agents, OPEN technique “Intelligent agent identification” can be applied (although it still requires enhancement). The determination of agents’ activities can be assisted by various Agent OPEN techniques “Commitment management”, “Activity scheduling”, “Task selection by agents”, “Deliberative reasoning”, and “Reactive reasoning”.

5.2.6. For Defining Application-Specific Ontologies

Description: An ontology can be formed by mapping logical predicates to domain relations, for example, predicate *position(?lat, ?long)* can be used to represent a user’s position in latitude and longitude. No techniques are provided on how to identify domain concepts/relations

Support from OPEN: Technique “Domain analysis” of OPEN can be used to support domain concepts identification.

5.2.7. For Building Agent Components

Description: Perceptor and actuator units can be identified by reviewing activities specified in the Activity Model. Perceptor units can be implemented as Java classes that encapsulate sensing abilities and convert raw data into beliefs. Actuator units can be realized as Java classes that contain actions directly executable by the agents.

Support from OPEN: Agent OPEN technique “Environmental evaluation” can be applied to determine how, and what sensing/affecting abilities are required for, the agents to interact with the environment.

5.2.8. For Building Platform Services

Description: FIPA-standards can be investigated to identify and build necessary platform services.

Support from OPEN: No techniques are found from OPEN that explicitly support the identification and design of infrastructure facilities. This is a topic for future research.

5.2.9. For Implementing Agent Classes

Description: Each agent class is implemented as a mental entity with beliefs (i.e. knowledge about the current state of itself and its environment), commitments (i.e. current and future activities that the agent has decided to perform), and commitment rules (i.e. mappings between beliefs and commitments).

transformation of agent class design to implementation, including “3-layer BDI model”, “deliberative reasoning: plans”, “reactive reasoning: ECA rules”, “Commitment management”, “activity scheduling”, “task selection”, and “belief revision”.

5.2.10. For Testing

Description: No techniques are provided for the formulation and execution of protocol and behaviour tests.

Support from OPEN: Conventional testing techniques “Unit testing” and “Integration testing” of OPEN can be extended to cater for agent behaviors testing and agent interactions testing respectively.

5.2.11. For Deployment

Description: Platform configuration file(s) need to be generated, specifying which agents should be initiated, which resources should be connected/created, and which facilities need to be used.

Support from OPEN: This is a topic for future research.

5.3. Work Products of the Agent Factory

The Agent Factory specifically aims, where possible, to use pre-existing design notations (Collier et al., 2004). Thus, Agent Factory adapts UML and Agent UML diagrams for its work products. The adaptations are outlined as follows:

5.3.1. System Behaviour Model

The system behaviour model is documented by using UML Use Case Diagrams where actors represent the roles to be played by agents (denoted as «role» stereotyped entities), and use cases represent behaviours associated with roles (denoted as “role-use-case” stereotyped entities).

5.3.2. Interaction Model

An interaction model uses UML Collaboration Diagrams to model “interaction scenarios”. Interacting objects represent roles (denoted with “role” stereotype), and message types are FIPA-ACL performatives (denoted with “fipa-acl” stereotype).

5.3.3. Activity Model

An activity model in the Agent Factory is depicted with regular UML Activity Diagrams to model “activity scenarios”, with each swimlane representing the processing of a role involved in the scenario.

5.3.4. Protocol Model

The Agent Factory protocol model is depicted with an Agent UML Sequence Diagram, one for each protocol (Figure 2). There may be secondary sequence diagrams which model protocol templates.

Figure 2. Example protocol model (adapted from Collier et al., 2004).

5.3.5. Agent Model

An agent model in Agent Factory is depicted using a UML Class Diagram that shows all agent classes in the system and their associated roles. Each role class is characterized by its associated protocols, while each agent class is characterized by a list of protocols (not those specified in roles) and activities.

6. SUMMARY AND ACKNOWLEDGEMENTS

A method engineering-based approach is, by its nature, advantageous to a single methodology “by itself”, since the ME-based approach encompasses the specific methodology *plus* an arbitrarily wide range of additional method fragments that can be combined with the original value, thus adding value to it.

As part of an extensive research programme to combine the benefits of method engineering and to extend an existing object-oriented framework (OPEN) to create a highly supportive methodological environment for the construction of agent-oriented information systems, we have analysed here contributions from the Agent Factory AO methodology. We have identified three new subtasks for pre-existing Tasks only. All other aspects of Agent Factory can be satisfactorily simulated using method engineering with OPEN. This means that a process engineer or project manager wishing to use the style of development advocated by the Agent Factory need only take the OPEN repository, as enhanced here, and select from it the appropriate method fragments from which to “method engineer” this particular agent-oriented approach to information systems development.

We wish to acknowledge financial support from the University of Technology, Sydney under their Research Excellence Grants Scheme. This is Contribution number 04/21 of the Centre for Object Technology Applications and Research.

REFERENCES

- Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A., 2004, Tropos: an agent-oriented software development methodology, *Autonomous Agents and Multi-Agent Systems* **8**(3):203–236.
- Brinkkemper, S., 1996, Method engineering: engineering of information systems development methods and tools, *Inf. Software Technol.* **38**(4):275–280.
- Brinkkemper, S., Saeki, M., and Harmsen, F., 1998, Assembly techniques for method engineering, *Procs. CAISE 1998*, LNCS1413, B. Pernici and C. Thanos, eds., Springer Verlag, pp. 381–400.
- Cockburn, A. S., 2000, Selecting a project’s methodology, *IEEE Software* **17**(4):64–71.
- Collier, R., et al., 2003, Beyond prototyping in the factory of agents, in: *Multi-Agent Systems and Applications III*, LNCS 2691, V. Marik, J. Muller, and M. Pechoucek, eds., Springer-Verlag, New York, pp. 383–393.

- Collier, R., O'Hare, G. and Rooney, C., A UML-based software engineering methodology for Agency Factory, Procs. SEKE 2004, Knowledge Systems Institute, Skokie, IL, USA (in press).
- Debenham, J., and Henderson-Sellers, B., 2003, Designing agent-based process systems – extending the OPEN Process Framework, Chapter VIII in: *Intelligent Agent Software Engineering*, V. Plekhanova, ed., Idea Group Publishing, Hershey, USA, pp. 160–190.
- FireSmith, D. G., and Henderson-Sellers, B., 2002, *The OPEN Process Framework. AN Introduction*, Addison-Wesley, Harlow, Herts, UK.
- Graham, I., Henderson-Sellers, B., and Younessi, H., 1997, *The OPEN Process Specification*, Addison-Wesley, Harlow, UK.
- Henderson-Sellers, B., 2003, Method engineering for OO system development, *Comm. ACM* **46**(10):73–78.
- Henderson-Sellers, B., and Debenham, J., 2003, Towards OPEN methodological support for agent-oriented systems development, in: *Procs. First International Conference on Agent-Based Technologies and Systems*, B. H. Far, S. Rochefort, and M. Moussavia, eds., University of Calgary, Canada, pp. 14–24.
- Henderson-Sellers, B., Giorgini, P., and Bresciani, P., 2003, Evaluating the potential for integrating the OPEN and Tropos metamodels, in: *Procs. SERP '03*, B. Al-Ani, H. R. Arabnia, and Y. Mun, eds., CSREA Press, Las Vegas, USA, pp. 992–995.
- Henderson-Sellers, B., Debenham, J., and Tran, Q. N. 2004a, Adding agent-oriented concepts derived from Gaia to Agent OPEN, in: *Advanced Information Systems Engineering. 16th International Conference, CAISE 2004, Riga, Latvia, June 2004 Proceedings*, LNCS 3084, A. Persson and J. Stirna, eds., Springer-Verlag, Berlin, pp. 98–111.
- Henderson-Sellers, B., Giorgini, P., and Bresciani, P., 2004b, Enhancing Agent OPEN with concepts used in the Tropos methodology, in: *Procs. ESAW'03 (Engineering Societies in the Agents World)*, LNAI Volume 3071, A. Omicini, P. Pettra, and J. Pitt, eds., Springer-Verlag, Berlin.
- Henderson-Sellers, B., Debenham, J., and Tran, N., 2004c, Incorporating elements from the Prometheus agent-oriented methodology in the OPEN Process Framework, in: *Procs. AOIS@CAISE'04*, Faculty of Computer Science and Information, Riga Technical University, Latvia, pp. 370–385.
- Henderson-Sellers, B., Tran, Q.-N. N., and Debenham, J., 2004d, Method engineering, the OPEN Process Framework and Cassiopeia, in: *The Symposium on Professional Practice in AI*, E. Mercier-Laurent and J. Debenham, IFIP, pp. 263–272.
- Klooster, M., Brinkkemper, S., Harmsen, F., and Wijers, G., 1997, Intranet facilitated knowledge management: A theory and tool for defining situational methods, in: *Procs. CAISE 1997*, LNCS1250, A. Olive and J. A. Pastor, eds., Springer Verlag, pp. 303–317.
- Luck M., Ashri, R., and D'Inverno, M., 2004, *Agent-Based Software Development*, Artech House, Boston, pp. 208.
- Nguyen, V. P., and Henderson-Sellers, B., 2003, OPENPC: a tool to automate aspects of method engineering, in: *Procs. ICSSEA 2003*, Paris, France, Volume 5, pp. 7.
- Ralyté, J., and Rolland, C., 2001, An assembly process model for method engineering, in: *Advanced Information Systems Engineering*, LNCS2068, K. R. Dittrich, A. Geppert, and M. C. Norrie, eds., Springer, Berlin, pp. 267–283.
- Rolland, C., and Prakash, N., 1996, A proposal for context-specific method engineering, in: *Procs. IFIP WG8.1 Conf. on Method Engineering*, S. Brinkkemper, K. Lytinen, and R. Welke, eds., Chapman and Hall, pp. 191–208.
- Rupprecht, C., Funffinger, M., Knublauch, H., and Rose, T., 2000, Capture and dissemination of experience about the construction of engineering processes, in: *Procs. 12th Conference on Advanced Information Systems Engineering (CAISE)*, LNCS 1789, B. Wangler and L. Bergman, eds., Springer-Verlag, Berlin, pp. 294–308.
- Saeki, M., 2003, CAME: the first step to automated software engineering, Process Engineering for Object-Oriented and Component-Based Development, in: *Procs. OOPSLA 2003 Workshop*, C. Gonzalez-Perez, B. Henderson-Sellers, and D. Rawsthorne, eds., Centre for Object Technology Applications and Research, Sydney, Australia, pp. 7–18.
- Ter Hofstede, A. H. M., and Verhoef, T. F., 1997, On the feasibility of situational method engineering, *Information Systems* **22**:401–422.
- Tran, Q. N., Henderson-Sellers, and B., Debenham, J. 2004, Incorporating the elements of the MASE methodology into Agent OPEN, in: *Procs. ICEIS2004*, I. Seruca, J. Cordeiro, S. Hammoudi, and J. Filipe, eds., INSTICC Press, Volume 4, pp. 380–388.
- Wooldridge, M., Jennings, N. R., and Kinny, D., 2000, The Gaia methodology for agent-oriented analysis and design, *Autonomous Agents and Multi-Agent Systems* **3**:285–312.