Incorporating Elements from the Prometheus Agent-Oriented Methodology in the OPEN Process Framework

B. Henderson-Sellers, Q.-N. N. Tran and J. Debenham

Faculty of Information Technology
University of Technology, Sydney
PO Box 123, Broadway
NSW 2007, Australia
{brian@it.uts.edu.au; numitran@yahoo.com; debenham@it.uts.edu.au}

Abstract. As part of an extensive research programme to combine the benefits of method engineering and existing object-oriented frameworks (notably the OPF) to create a highly supportive methodological environment for the construction of agent-oriented information systems, we have analysed ~~here~~ contributions to the OPF repository of process components from the Prometheus agent-oriented methodology. We have identified three new Tasks, together with two new subtasks for a pre-existing Task and one additional Technique. Prometheus has also supplied the OPF with four new Work Products but no additional Roles or Stages.

1 Introduction

The increasing interest in agents and agent-oriented methodologies requires the construction of high quality software applications. There are an increasing number of stand-alone methodologies but none of these supports all methodology elements across the full lifecycle, instead focussing solely on agent-specific issues such as social interaction, autonomy and reasoning processes. There is also debate as to whether an agent-oriented methodology should be seen as a new mindset requiring a brand new approach or whether it can be considered as extending the object-oriented paradigm and methodology. While the former view has been advocated by e.g. Tropos [1], we have already shown [2,3] that this view can be accommodated within an object-oriented framework approach and it is this view that is taken in this paper.

Object-oriented (OO) methodologies do not take into account agency concepts. Recently there has been interest in identifying how an object-oriented methodology might be extended or enhanced to support these newer ideas. Several proposals have been made. For instance, Gaia [4] takes as its basis the Fusion methodology [5]; ADELFE [6] starts with RUP [7] (although the citation is actually to the Unified Software Development Process [8]).Such methodologies have a specific and restricted focus, for instance to a particular lifecycle stage e.g. Tropos [1] focusses on early requirements engineering. Outside their stated scope, however, they have little or no application. Here, we focus on a *comprehensive* software development approach and seek a methodological environment that has the capacity to be flexible enough to

support new ideas as they appear and become accepted as well as the old, well-established ones [9]. Merging of methodologies has been addressed from an informal viewpoint [10]; however, a more rigorous and highly promising approach to offer such support is that of method engineering [11,12].

This paper reports on part of an extensive research programme to combine the benefits of method engineering and existing object-oriented frameworks (notably the OPEN Process Framework or OPF [13,14]) to create a highly supportive environment for the construction of agent-oriented information systems. By starting with a repository of OO method fragments, the research questions relate to the identification of new (or extended) process components necessary to support various flavours of agent-oriented applications development. To do this, we are examining each of the extant AO methodologies in turn to see what must be added to the OPF repository so that that particular AO methodology can be (re)created from the elements in the extended OPF. In this paper, we focus on the Prometheus AO methodology [15,16] to complement our previous analysis of, *inter alia*, Tropos [3], Gaia [17] and MaSE [18].

In Section 2, we outline the ideas of method engineering (ME) followed by a brief summary in Section 3 of our selected OO situational method engineering approach – that based on the OPF [14]. In Section 4, we describe the basics of Prometheus and then in Section 5 describe the elements of Prometheus not currently supported in the OPF and which we therefore propose for addition to the OPF repository.

2 Method Engineering

Method engineering [11] is a rational approach to the construction, either fully or partially, of methods (a.k.a. methodologies) from method fragments (often called method chunks [19] or process components[8] [14]), typically stored in a repository. The method itself is constructed by selection of appropriate method fragments followed by their configuration in such a way as to satisfy the requirements for the method [20] and create a meaningful overall method [21]. A method that is targetted at a particular project or environment is known as a situated or situational method and the means of its derivation known as situational method engineering (SME) [12].

Ideally, a method engineering approach to process construction will utilize a process metamodel from which current and future method fragments can be generated by the instantiation rule. These generated fragments will be consistent with the rules of the metamodel, a rule for instance that might state that a producer (usually a person) can utilize a work unit (such as a task) in order to product some kind of work product (e.g. documentation, code). Such rules also automatically impose some granularity constraints as noted in [21]. A second set of rules and guidelines is needed to assist in process construction [22], an approach that can also potentially be automated [23,24]. Here, we call a combination of metamodel and generated process components (stored in a repository) a process framework

[8] A methodology is a combination of a process and a set of products. We focus on the process portion. Thus, the words "method", "methodology", "process" are taken here as synonyms.

3 The OPEN Process Framework

The OPEN (Object-oriented Process, Environment, and Notation) Process Framework (OPF) [14] combines a process metamodel and a repository of process components (Figure 1). Elements from the repository are selected and put together to form a specific process or situational method. Process elements are related to other process elements with a possibility value known as a deontic value [13]. Deontic matrices can be defined for (1) Process/Activity, (2) Activity/Task, (3) Task/Technique, (4) Producer/Task, (5) Task/Work Product, (6) Producer/Work Product and (7) Work Product/Language. Deontic values have one of five values ranging from mandatory through optional to forbidden. This gives a high degree of flexibility to the process engineer who can allocate different deontic values for any specific pair of process components depending upon the context i.e. the specific project, skills set of the development team etc. Allocating these deontic values is the responsibility of the process engineer, perhaps assisted by an automated tool along the lines recently proposed [23]. This deontic approach to process construction is one of OPEN's strengths that make it suitable for a wide range of project types.

When used on a specific project, this is known as a process instance. A company-customized OPEN version is then "owned" by the organization – it is their own internal standard, yet retains compatibility with the global OPEN user community.

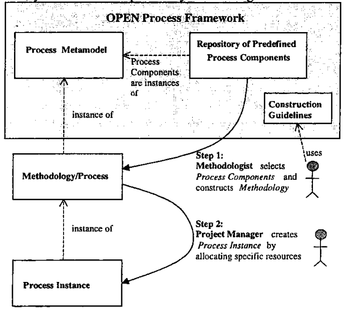
OPEN Process Framework
Process Metamodel
Repository of Predefined Process Components
Process Components are instances of
Construction Guidelines
instance of
uses
Methodology/Process
Step 1: Methodologist selects Process Components and constructs Methodology
instance of
Step 2: Project Manager creates Process Instance by allocating specific resources
Process Instance

Fig. 1. The OPF defines a framework consisting of a metamodel and a repository of process components. Construction guidelines are used to create a site-specific or project-specific methodology/process, which can then be enacted (as a "process instance") on a specific project (after [25])

3.1 The OPF Metamodel

The OPF metamodel defines five main high level classes of process components:

Work Product: "is anything of value that is produced during the development process" [14]. Work Products are the result of producers (people) executing Work Units and are used either as input to other Work Units or delivered to a client. Pragmatically, they also include externally-supplied (e.g. by user) pre-existing artifacts used as inputs to Work Units.

Producer: "is responsible for creating, evaluating, iterating and maintaining Work Products" [14].

Work Unit: a functionally cohesive operation that is performed by a Producer. There are three major classes of Work Unit: Activity, Task and Technique.

Language: a medium for documenting a Work Product.

Stage: an identified and managed duration within the process or a point in time at which some achievement is recognized.

Each of these metaclasses has many subclasses in the detailed metamodel (see Appendix G of [14]). From each of these subclasses one or more process component instances are generated and stored in the OPF repository (Figure 1)

3.2 Process Components in the Repository

Initially, the OPF repository contained about 30 predefined instances of Activity, 160 instances of Task and 200 instances of Techniques (the three main kinds of Work Unit) as well as multiple instances of Role, Stage, Language etc. Some of these are orthogonal to all others in their group and some overlap. Consequently, during process construction both association and integration strategies [20] are needed. For example, there are several Techniques in the repository for finding objects e.g. textual analysis, use cases simulations, CRC card techniques.

The Work Units in the OPF are perhaps the most obvious during the initial stages of process construction. At the highest abstraction level are Activities which describe what needs to be undertaken. The overall software development process is often configured using half a dozen or so of these Activities so they are often (but not always) the first to be identified. Tasks are then added which give more detail but still focus on "what" needs to be done rather than "how" it is do be done. Tasks can be readily tracked and project managed. They are typically allocated to a small team over a period of a few days. Scheduling comes from the planning Activities and Tasks combined with the project management elements embodied in the appropriate Tasks. Accomplishment of Tasks is by the use of appropriate Techniques. This set of process components is diverse since there are often parallel Techniques for any given Task.

Work Products, including models, documentation and metrics, are also important in any software development process – although a methodology that is driven by the desire to produce documentation often fails because it delivers the Work Products for their own sake. Care must therefore be taken in selection of appropriate process components for Work Products.

Producers bring in the human element (although some producers may be other software or indeed hardware). Producers play various roles within the methodology. It is important to identify these roles rather than the people playing them. A large number of these are described in the OPF repository although it is a volatile set in comparison with, say, the instances of the WorkUnit class.

Since its first publication in 1997, several additions have been made to the OPF repository to enhance its support for

- Web development [26,27]
- Component-based development [28]
- Organizational transition [29,30]
- Usage-centered design [25]

As well as some initial work on agent extensions [3,31].

In Section 5 of this paper, we extend the OPF repository even further, to offer additional support for agent orientation (AO) by extracting new process components from the Prometheus AO methodology [15,16].

4 Major Elements of Prometheus

Prometheus [15,16] is an agent-oriented methodology with three design phases ending in an implementation phase. First is the systems specification phase in which the basic functionality of the system is identified, using percepts (inputs), actions (outputs) and any necessary shared data storage. This is followed by the architectural design stage which uses as input the outputs of the previous phase. Here, the agents and their interactions are identified. Finally, there is the detailed design phase in which the internal details of each agent are addressed.

Prometheus reuses as much as possible from object technology. In particular, it uses UML sequence diagrams (as Prometheus's interaction diagrams) and UML use cases form the basis for the Prometheus variant named scenario.

Within each of these three major Prometheus phases, we have identified a number of tasks, together with advice on appropriate techniques and work products generated or consumed. These are detailed in the following subsections, together with a brief comment on the stages (Prometheus phases) and languages recommended for use with the Prometheus approach to building agent-oriented software applications.

4.1 Tasks Characterizing Prometheus

- ***'Identifying Percepts and Actions'*:** "Percepts" are raw data obtained from the environment, while "actions" are agents' mechanisms for affecting the environment. This task determines how agents interact with their environment.
- ***'Identifying System Goals and Functionality'***: This task aims to determine what the target system should do in a broad sense. Each goal is associated with a (set of) functionality. Each functionality is described in terms of various attributes (see Functionality Descriptor Work Product in Section 4.3 for more details).
- ***'Specifying Use Case Scenarios'***: This task aims to give a more holistic view of the system processing (as compared to Functionality Specification which focusses on particular aspects of the system). This task also helps to identify further functionality which may otherwise be missed.
- ***'Identifying Agent Types and Instances':*** Agent types are identified from functionality (by grouping closely-related functionality into one agent type). For each agent type, the designer needs to determine the number of agent instances,

the lifetime of each instance, agent initialisation, agent demise, data used/produced and the events with which the agent will deal.
- ***'Identifying Events'***: Events are derived from percepts. These events are things that the agent will notice, which will cause it to react in some way.
- ***'Identifying and Specifying Shared Data Objects'*:** If multiple agents write to the same shared data objects, this will require significant additional care for synchronization. At this step, the designer should also decide the appropriate data-sharing mechanism (e.g. having one agent managing the shared data object, or having each agent storing its own version of the information). This step also helps to evaluate design, as a good design will minimize shared data objects.
- ***'Specifying Agent Interaction'***: This task involves developing Interaction Diagrams to show the major sequences of interactions between agent types, and secondly, Interaction Protocols to elaborate each interaction (as shown in Interaction Diagrams) to show all potential variations.
- ***'Designing Agent Internal Structure'***: This task involves a progressive refinement process in order to define the structure of each agent type by defining and then elaborating agent *capabilities*. These are refined in turn until all capabilities have been defined. At the bottom level, capabilities are defined in terms of plans, internal events and data.

4.2 Techniques Recommended by Prometheus

- ***For 'Identifying Percepts and Actions'*:** no specific techniques for identifying Percepts and Actions are identified in the Prometheus documentation.
- ***For 'Identifying System Goals and Functionality'***: Functionality should be kept as narrow as possible, dealing with a single aspect or sub-goal of the system. If functionalities are too broad, they are likely to be less adequately specified leading to potential misunderstanding. Each functionality should be linked to some System Goal, while each goal should result in one or more functionality, although there may not be a one-to-one mapping. The identification of functionality can be performed in conjunction with Use Case Scenarios specification: typically some functionalities are defined, these being used to specify use case scenarios, which in turn identify more functionalities.
- ***For 'Specifying Use Case Scenarios'*:** The central part of a use case scenario is a sequence of steps describing the scenario. Each step should be annotated with the name of the functionality responsible and data read/written. These annotations allow for cross checking with the functionality description. The final set of use cases should provide a good overview of how the system will work.
- ***For 'Identifying Agent Types and Instances'***: Functionalities are assigned to agents based on the criteria of *strong coherence* and *loose coupling*. Specifically, they are grouped to a single agent if they are related, use the same data and interact frequently with each other. Reasons against groupings include when functionalities are unrelated, exist on different hardware platforms or when there exist security, privacy and modifiability issues. A simple heuristics test of whether a suitable name for an agent type can be found is useful for evaluating coherence. A coherent agent should be able to be described by a single term without any conjunctions. Data Coupling Diagrams and Agent

Acquaintance Diagrams can be used to determine and evaluate the potential agent groupings. A design with an Acquaintance Diagram where each agent is linked to every other agent is undesirable.

- ***For 'Identifying Events'***: Events should be generated as a result of percepts from the environment, either directly or after processing. The designer should look for changes between the current and previous percepts, or between believed states of world and percepts. Events can be externally generated (from percepts) or internally generated (from messages sent from one agent to another).
- ***For 'Identifying and Specifying Shared Data Objects'***: Often what at first appears to be a shared data object can be reconceptualised to be a data source managed by a single agent, with information provided to other agents when needed. Alternatively, each agent may have its own version of the information, without there being any need for a single centralized data object. Data objects can be specified using traditional OO techniques or database design techniques.
- ***For 'Specifying Agent Interaction'***: Interaction Diagrams and Interaction Protocols need to be developed. Interaction diagrams are borrowed directly from OO design, showing interaction between agents rather than objects. Designers can directly use the use case scenarios developed earlier to build corresponding Interaction Diagrams. Wherever there is a step in the use case that involves functionality from a new agent, there must be some interaction from a previously involved agent to the newly participating agent. Also, each major environmental event should have an associated Interaction Diagram. Interaction Protocols are generalizations of Interaction Diagrams. They define precisely which interaction sequences are valid within the system. Since protocols must show all variations, they are often larger than the corresponding Interaction diagram.
- ***For 'Designing Agent Internal Structure'***: Functionality (identified from the task 'Identifying System Goals and Functionality') provides a good initial set of capabilities. Sometimes functionality is required in multiple places - akin to library routines. Such functionality should also be extracted into a capability which can then be included in other capabilities or agents. Each capability is composed of input/output events, data read/written, plans and sub-capabilities.

4.3 Work Products Advocated by Prometheus

- ***Functionality Descriptor***: This is a textual template which describes each functionality in terms of name, description, percepts, actions, data used/produced and a brief discussion of interactions with other functionality.
- ***Use Case Descriptor***: This is a textual template which describes the sequence of steps involved in each use case scenario. Each step is either an incoming event/percept, message, action or activity (activity is anything *within* the functionality, e.g. some kind of computation). All of these elements can be derived from Functionality Descriptors.
- ***Agent Class Descriptor***: This is a textual template that describes each agent in terms of functionality included, data used/produced, incoming events, actions, lifetime, initialisation, demise and other agents with which it interacts.

- ***Agent Acquaintance Diagram:*** This diagram type (Figure 2) shows undirected communication links between agent types. It is developed during the 'Identifying Agent Types and Instances' task to assist 'Agent Type Identification'.

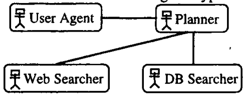

Fig. 2. Example of an Agent acquaintance diagram

- ***System Overview Diagram:*** This diagram provides a general understanding of how the system functions as whole. It shows agents, events, shared data objects and interactions between agents (Figure 3).

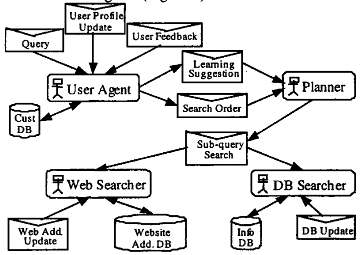

Fig. 3. Example of System Overview diagram

- ***Agent Interaction Model***: including Interaction Diagrams and Interaction Protocols - both diagrams expressed with AUML Sequence Diagrams (Figure 4).
- ***Agent Overview Diagram***: This diagram provides the top level view of the agent internals. It shows the top level capabilities of the agent, events or task flows between these capabilities, as well as data internal to the agent (Figure 5). Further levels of details will be provided by Capability Diagrams.
- ***Capability Diagram***: Each Capability Diagram describes an agent's capability in details. At the bottom level, this diagram contains plans, internal events (which connect plans) and data used/produced by plans. Capability Diagrams are similar in style to System Overview (Figure 3) and Agent Overview Diagram (Figure 5), although plans are constrained to have a single incoming event.

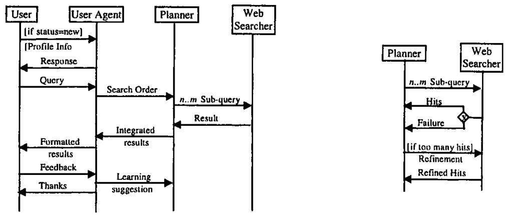

Fig. 4. Example of Interaction and protocol diagrams

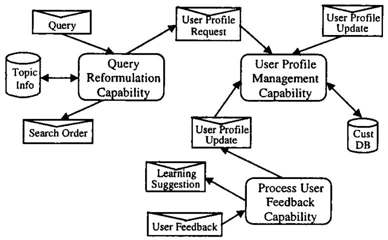

Fig. 5. Example of Agent Overview diagram (for User Agent)

- ***Plan, Event, Data Descriptor***: Each Plan Descriptor defines a plan in terms of triggering events, messages, actions, and plan steps. Each Event Descriptor defines an event in terms of the event's purpose, together with data that the event carries. Each Data Descriptor defines a data object in terms of its fields and methods. All three descriptors are expressed as textual templates.

4.4 Stages used in Prometheus

Cycle: Prometheus uses an iterative process over software engineering phases, thus fitting the "*Iterative, Incremental, Parallel Life Cycle*" model of OPEN.

Phases: Prometheus covers System Specification, Architectural Design, and Detailed Design. Outputs of these phases can be directly fed into implementation and testing. In the context of OPEN, Prometheus supports "*Initiation*" and "*Construction*".

4.5 Languages

For modelling, Prometheus proposes its own set of notation, except for the Interaction Model, which employs the AUML Sequence Diagram [32]. For an implementation language, it is noted that Prometheus models can be implemented in any programming language although exemplar implementations have been undertaken using the JACK Development Platform.

5 Adding support to the OPF derived from Prometheus

In this section, we outline the various Tasks, Techniques and Work Products that are proposed in this paper as additions and modifications to the OPEN repository in order to incorporate agency concerns as identified in Prometheus. These new process components are identified from the literature and proposed here for inclusion into the OPF process component repository.

In total, three new Tasks are identified, together with two new subtasks for a pre-existing Task. An additional Technique, suitable for agent support, has also been identified for inclusion in the OPF repository (and one that will need further extension - not discussed in detail here). Prometheus supplies a total of 5 new Work Products but no additional Roles or Stages.

Many of the tasks described in Prometheus are already incorporated into the process component library of OPEN, albeit sometimes under a different name. First we discuss the mappings to existing process components (Section 5.1) and then identify new components for addition to the OPF repository (Sections 5.2 to 5.4).

5.1 Existing support and mappings between OPF and Prometheus

The Prometheus task of 'Identifying Percepts and Actions' maps to the Agent OPEN Tasks: 'Model the agent's environment'. Useful techniques here might be 'Context Modelling', 'CRC Card Modelling' and 'Event Modelling'. 'Identifying Systems Goals and Functionality' in Prometheus is paralleled by the set of Requirements Engineering tasks in OPEN that directly cover this Prometheus task (particularly OPEN's 'Identify Client's Vision' and 'Analyze Requirements'). The OPEN task: 'Use Case Modeling' corresponds directly to the Prometheus task of 'Specify Use Case Scenarios'.

The Prometheus task named 'Identifying Agent Types and Instances' leads to the need for a new OPF Task which we name 'Construct the Agent Model' (see Section 5.2). Similarly, Prometheus's 'Identifying and Specifying Shared Data Objects' identifies a gap which we fill with a new OPF Task: 'Specify shared data objects'. On the other hand, there is some high level support in Agent OPEN's Task: 'Model the agent environment' for both events and percepts. However, we recommend supporting these more substantially through the introduction of two new subtasks (see Section 5.2 for details).

For Prometheus's 'Specifying Agent Interaction' there is a good mapping to OPEN's Task: 'Construct the object model'/Technique: 'Interaction modelling' [14] plus Task: 'Determine agent interaction protocol' [31]; whereas the Prometheus's 'Designing Agent Internal Structure' requires the addition to the OPF repository of a new task called 'Design agent internal structure'.

There are many existing OPF Techniques covering those required by Prometheus. Techniques to support Prometheus's 'Identifying percepts and actions' include 'Context modelling', 'CRC card modelling' and 'Event modelling'. For 'Identifying System Goals and Functionality', OPF already has 'Hierarchical task analysis'. For 'Specifying Use Case Scenarios', OPF offers Technique: 'Scenario development'.

For 'Identifying Agent Types and Instances' in Prometheus, the OPF has some support, although inadequate in parts. While existing Techniques of 'Cohesion measurement' and 'Coupling measurement' offer support, the existing Technique: 'Intelligent agent identification' covers only the need for agents and agent modelling notation and significant extension will be required. OPEN also offers various techniques for OO class identification/modelling (such as 'Abstract Class Identification' and 'Class Naming'), which can be extended/adapted for the identification of agent classes. The extension should take into account the major differences between OO classes and agent classes, for example, agent classes are generally more coarse-grained than OO classes (thus, the 'Granularity' Technique in the OPF repository should be extended to account for this difference).

For 'Identifying Events' in Prometheus, the OPEN Technique: 'Event Modelling' is directly applicable. OPEN currently offers no techniques for the Prometheus task of 'Identifying and Specifying Shared Data Objects'. Support for 'Specifying Agent Interactions' is partial through the OPF Techniques of 'Interaction modelling' and 'Collaboration analysis'. However, these need extension. The Agent OPEN [31] Techniques of 'Contract nets' and 'Market mechanisms' may also be useful. Finally, we need to propose a new Technique (Section 5.3) to support the Prometheus task of 'Designing Agent Internal Structure'. This we name 'Agent internal design'.

For work products, Prometheus uses a suite of diagrams that include both new diagrams and extensions of existing (often UML) diagrams. We therefore propose the addition of four new diagram types to the OPF process component repository. Those that can be classified as belonging to the suite of OPF Static Architecture Diagrams are Agent Model; Agent Acquaintance Diagram; Agent Overview Diagram

The agent interaction model has two components: a standard Interaction Diagram and a new Protocol Diagram. Other new diagrams are the Functionality Descriptor and Capability Diagrams. The Capability Diagram is essentially the same as that proposed in Tropos – a diagram already been incorporated into the OPF repository [3]. The Agent Overview Diagram is essentially a Context Diagram so no new work product is proposed here for the OPF repository. Finally, there is a close mapping from the Prometheus Use Case Descriptor to the OPF Use Case Specification.

5.2 New Tasks

Although these tasks are a contribution to the OPF, commonly found in several AO methods, we itemize them here since they are currently missing from the repository.

TASK NAME: Construct the agent model
Focus: Static architecture

Typical supportive techniques: Intelligent agent identification, Control architecture
Explanation: An analogue of the "object model" as the main description of the static architecture needs to be constructed. This model shows the agents, their interfaces and how they are connected both with other agents and other objects.

TASK NAME: Design agent internal structure
Focus: Internal structure of agents
Typical supportive techniques: Agent internal design, 3-layer BDI model, Reactive reasoning
Explanation: Using an appropriate model for the internal agent architecture, such as the BDI model, the internal structure of each agent needs to be determined. If a hybrid architecture is used, then both ECA rules (event-condition-action rules) and I-rules (inference rules) may be needed. If using a BDI architecture, then goals and plans will be needed (see Agent OPEN Tasks: 'Model goals' and 'Model plans': [3]. When using Prometheus, high level capabilities are identified and iteratively decomposed, finally resulting in plans, internal events and data.

TASK NAME: Specify shared data objects
Focus: Data storage
Typical supportive techniques: appropriate database-focussed techniques
Explanation: Synchronization is required if several agents write to the same data storage object. Appropriate data-sharing mechanisms are needed.

Finally, two subtasks are recommended for addition to the existing Task: Model the Agent's Environment

Subtask: Model Percepts. This task focuses on modelling the percept component of the agent's environment.

Subtask: Model Events. This task focuses on modelling the events that result from changes in the environment which are then recognized by the agent itself as an input to its own internal reasoning.

5.3 New Technique

Although this is a contribution common to many AO methods, we itemize it here in the context of it being currently missing from the OPF repository.

TECHNIQUE NAME: Agent internal design
Focus: Internal features of an agent
Typical tasks for which this is needed: Design agent internal structure
Technique description: The fine detail of an individual agent must be described in terms of its attributes and operations (as for objects) but more importantly in terms of its goals, plans, capabilities, responsibilities, events responded to and pre- and post-conditions.
Technique usage: Document each of these internal characteristics (or features) of every agent in the system. The detail should be sufficient for coding to take place easily from these design specifications.
Deliverables: Capability diagram

5.4 New Work Products

Although these Work Products are not unique to Prometheus, they are currently missing from the OPF repository and are therefore documented here.

NAME: Functionality Descriptor
OPF CLASSIFICATION: Requirements set of work products
RELATIONSHIP TO EXISTING WORK PRODUCT: None
BRIEF DESCRIPTION: This is a textual template describing the functionality in terms of name, description, percepts, actions, data used/produced and a brief discussion of interactions with other functionality.

NAME: Agent structure diagram
OPF CLASSIFICATION: Static Architecture diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT: extension of static structure diagram
BRIEF DESCRIPTION: This diagram describes the internal structure of an individual agent. It needs to explain how inputs are received (events), thus linking to the Functionality Descriptor, what goals and plans are possessed and how reasoning is accomplished. If using a BDI model for the agent architecture, then an extension to the UML class notation might look like the proposal in Figure 6 in which the additional boxes already supported in the UML are utilized here for agent, rather than object, concepts.

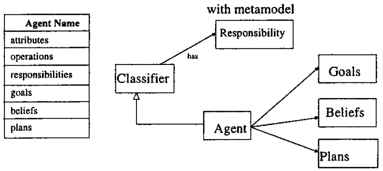

Fig. 6. Proposal for extended UML notation for an individual agent plus the underpinning metamodel fragment

NAME: Agent acquaintance diagram
OPF CLASSIFICATION: Static Architecture diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT: modification of a collaboration diagram
BRIEF DESCRIPTION: The agent acquaintance diagram is a simplified version of a UML V1.4 collaboration diagram. It shows the agents and connectivity without necessarily specifying directionality of connections.

NAME: Agent protocol diagram
OPF CLASSIFICATION: Dynamic Behaviour diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT: None

BRIEF DESCRIPTION: Agent protocols are shown using a UML Sequence Diagram. This diagram complements a standard sequence diagram to show agent interactions. Together, these two diagrams constitute the Agent Interaction Model.

NAME: Agent overview diagram
OPF CLASSIFICATION: Static Architecture diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT: extension of UML object model
BRIEF DESCRIPTION: A high level diagram showing capabilities of the agents, events between these capabilities and any internal data. This top level diagram forms the basis for further expansion (addition of detail) into a number of Capability Diagrams [1,3].

6 Summary and conclusions

As part of an extensive research programme to combine the benefits of method engineering and existing object-oriented frameworks (notably the OPF) to create a highly supportive methodological environment for the construction of agent-oriented information systems, we have analysed here contributions from the Prometheus AO methodology. We have identified three new Tasks, together with two new subtasks for a pre-existing Task. One additional Technique has also been identified for inclusion in the OPF repository plus one additional Technique that will need further extension – not discussed in detail here. Prometheus has also supplied the OPF with four new Work Products but no additional Roles or Stages.

As part of an ongoing projects, we are analysing individual AO methodologies to identify missing fragments prior to undertaking a full integration across *all* AO methodologies where we will identify any overlaps and omissions.

Acknowledgements

We wish to acknowledge financial support from the University of Technology, Sydney under their Research Excellence Grants Scheme. This is Contribution number 04/03 of the Centre for Object Technology Applications and Research.

References

1. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopolous, J. and Perini, A., 2003, Tropos: an agent-oriented software development methodology, *Autonomous Agents and Multi-Agent Systems,* 8(3), 203-236
2. Henderson-Sellers, B., Giorgini, P. and Bresciani, P., 2003, Evaluating the potential for integrating the OPEN and Tropos metamodels, *Procs. SERP '03* (eds. B. Al-Ani, H.R. Arabnia, and Y.Mun), CSREA Press, Las Vegas, 992-995
3. Henderson-Sellers, B., Giorgini, P. and Bresciani, P., 2004, Enhancing Agent OPEN with concepts used in the Tropos methodology, *Procs. ESAW'03 (Engineering Societies in the Agents World),* London, October 29-31 (in press)

4. Wooldridge, M., Jennings, N.R. and Kinny, D., 2000, The Gaia methodology for agent-oriented analysis and design, *J. Autonomous Agents and Multi-Agent Systems,* **3**, 285-312
5. Coleman, D., Arnold, P., Bodoff, S., Dollin, C. and Gilchrist, H., 1994, *Object-Oriented Development. The Fusion Method*, Prentice Hall, Englewood Cliffs, NJ, USA, 313pp
6. Bernon, C., Gleizes, M.-P., Picard, G. And Glize, P., 2002, The ADELFE methodology for an intranet system design, presented at AOIS2002, Toronto, 27-28 May
7. Kruchten, Ph., 1999, *The Rational Unified Process. An Introduction*, Addison-Wesley, Reading, MA, USA
8. Jacobson, I., Booch, G. and Rumbaugh, J., 1999, *The Unified Software Development Process*, Addison-Wesley, Reading, MA, USA
9. van Slooten, K., Hodes, B., 1996, Characterizing IS development projects, **in** *Procs. IFIP TC8 Working Conf. on Method Engineering: Principles of method construction and tool support* (eds. S. Brinkkemper, K. Lyytinen, R. Welke) Chapman&Hall, Great Britain, 29-44
10. Juan, T., Sterling, L., Martellis, M. And Mascardi, V., 2003, Customizing AOSE methodologies by reusing AOSE features, *Procs. AAMAS '03*, ACM Press, 113-120.
11. Brinkkemper, S., 1996, Method engineering: engineering of information systems development methods and tools, *Inf. Software Technol.*, **38(4)**, 275-280.
12. Ter Hofstede, A.H.M. and Verhoef, T.F., 1997, On the feasibility of situational method engineering, *Information Systems*, **22**, 401-422
13. Graham, I., Henderson-Sellers, B. and Younessi, H., 1997, *The OPEN Process Specification*, Addison-Wesley, UK.
14. Firesmith, D.G. and Henderson-Sellers, B., 2002, *The OPEN Process Framework. AN Introduction*, Addison-Wesley, Harlow, Herts, UK
15. Padgham, L. and Winikoff, M., 2002, Prometheus: A Methodology for Developing Intelligent Agents, **In** *Procs. Third International Workshop on Agent-Oriented Software Engineering, at AAMAS'02.*
16. Padgham, L. and Winikoff, M., 2002, Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents. **In** *Procs.Workshop on Agent-oriented Methodologies at OOPSLA 2002*, November 4, 2002, Seattle.
17. Henderson-Sellers, B., Debenham, J. and Tran, Q.-N.N., 2004, Adding agent-oriented concepts derived from GAIA to Agent OPEN, *Procs. CAiSE2004,* Springer-Verlag, Berlin, Germany
18. Tran, Q.-N.N., Henderson-Sellers, B. and Debenham, J. 2004a, Incorporating the elements of the MASE methodology into Agent OPEN, *Procs. ICEIS2004* (eds. I. Seruca, J. Cordeiro, S. Hammoudi and J. Filipe), INSTICC Press
19. Rolland, C. and Prakash, N., 1996, A proposal for context-specific method engineering, *Procs. IFIP WG8.1 Conf. on Method Engineering,* 191-208, Atlanta, GA, USA
20. Ralyté, J. and Rolland, C., 2001, An assembly process model for method engineering, in K.R. Dittrich, A. Geppert and M.C. Norrie (Eds.) *Advanced Information Systems Engineering*), LNCS2068, Springer, Berlin, 267-283.
21. Brinkkemper, S., Saeki, M. and Harmsen, F., 1998, Assembly techniques for method engineering. *Proceedings of CAISE 1998,* Springer Verlag, 381-400.
22. Rolland, C., Prakash, N. and Benjamen, A., 1999, A multi-model view of process modelling, *Requirements Eng. J.*, **4(4)**, 169-187
23. Nguyen, V.P. and Henderson-Sellers, B., 2003, Towards automated support for method engineering with the OPEN approach, *Procs. 7th IASTED SEA Conference*, ACTA Press, Anaheim, USA, 691-696
24. Saeki, M., 2003, CAME: the first step to automated software engineering, *Procs. OOPSLA 2003 Workshop on Process Engineering for Object-Oriented and Component-Based Development*, Centre for Object Technology Applications and Research, Sydney, Australia, 7-18

25. Henderson-Sellers, B. and Hutchison, J., 2003, Usage-Centered Design (UCD) and the OPEN Process Framework (OPF), *Performance by Design. Procs. forUSE2003* (ed. L.L. Constantine), Ampersand Press, Rowley, MA, USA, 171-196
26. Haire, B., Henderson-Sellers, B. and Lowe, D., 2001, 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, 383-389.
27. Henderson-Sellers, B., Haire, B. and Lowe, D., 2002, Using OPEN's deontic matrices for e-business, *Engineering Information Systems in the Internet Context* (eds. C. Rolland, S. Brinkkemper and M. Saeki), Kluwer Academic Publishers, Boston, USA, 9-30.
28. Henderson-Sellers, B., 2001, An OPEN process for component-based development, Chapter 18 in G.T. Heineman and W. Councill (Eds.) *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley, Reading, MA, USA, 321-340
29. Henderson-Sellers, B. and Serour, M., 2000, Creating a process for transitioning to object technology, *Procs. Seventh Asia-Pacific Software Engineering Conference. APSEC 2000,* IEEE Computer Society Press, Los Alamitos, CA, USA, 436-440.
30 Serour, M., Henderson-Sellers, B., Hughes, J., Winder, D. and Chow, L., 2002, Organizational transition to object technology: theory and practice, *Object-Oriented Information Systems* (eds. Z. Bellahsène, D. Patel and C. Rolland), LNCS 2425, Springer-Verlag, Berlin, 229-241.
31. Debenham, J. and Henderson-Sellers, B., 2003, Designing agent-based process systems - extending the OPEN Process Framework, Chapter VIII in *Intelligent Agent Software Engineering* (ed. V. Plekhanova), Idea Group Publishing, 160-190.
32. Odell, J., Van Dyke Parunak, H. and Bauer, B., 2000, Extending UML for agents. In G. Wagner, Y. Lesperance and E. Yu (eds.), *Procs. Agent-Oriented Information Systems Workshop,* 17th National Conference on Artificial Intelligence (pp. 3-17). Austin, TX, USA.