METHOD ENGINEERING, THE OPEN PROCESS FRAMEWORK AND CASSIOPEIA

B. Henderson-Sellers, Q.-N. N. Tran and J. Debenham
University of Technology, Sydney

Abstract: The Cassiopeia approach adopts an organization-oriented approach to multiagent systems design. In order to support the concepts in Cassiopeia using an existing process framework (the OPEN Process Framework or OPF), we identify two new Tasks, together with two new subtasks for a pre-existing Task and one additional Work Product, that need to be added to the existing OPF repository. Using method engineering it then becomes possible to generate a tailored agent-oriented methodology from this suite of process components.

Key words: Agent-oriented methodology, OPEN Process Framework, Cassiopeia

1. INTRODUCTION

Method engineering [1] provides a means of creating methods (or methodologies) that are constructed and tailored for specific situations. Indeed, a better name for method engineering is suggested as "situated method engineering" or SME [2]. 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 statements helps the method engineer to identify appropriate method components or method chunks [3] held in the SME repository and then to use construction guidelines [4,5] to finalize the highly specific methodology for use by the aforementioned organization.

Ideally, the elements in an SME repository should be compliant with (in fact generated from) a set of concepts described by a metamodel [6]. One such example is the metamodel+repository-based OPEN Process Framework or OPF [7]. The OPF 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 OPF 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 OPF repository for their potential support for the Cassiopeia agent-oriented approach to software development.

Since using a metamodel-based repository of method elements permits the construction of situated or individual methodologies, it is highly appropriate to ask if the existing OPF repository can support agent-oriented software development. Our project aims to ensure that adequate support is added to the OPF repository to enable this to occur successfully. Our approach is to analyze each existing AO methodological approach in turn, add any necessary method chunks to the repository and then, later, analyze potential conflicts. In this paper, we concentrate on the Cassiopeia approach for agent-oriented development [8,9]. In Section 2, we present an overview of Cassiopeia followed, in Section 3, by a detailed analysis of the Tasks, Techniques and Work Products required in Cassiopeia, evaluating whether the OPF already offers such support and, where not, what additional method chunks need to be added to the OPF repository.

2. BRIEF OVERVIEW OF CASSIOPEIA

Cassiopeia provides a (arguably incomplete) methodological framework for the development of collective problem-solving MASs, i.e. MASs where agents work together to achieve a collective task(s). Cassiopeia assumes that, although the agents can have different aims, the goal of the designer is to make them behave cooperatively. It adopts an *organization-oriented approach* to MAS design, as do some other AO approaches such as Gaia [10]. In other words, it views an MAS as an organization of agents that implement/encapsulate *roles*. These roles not only reflect the agents’ individual functionality, but also the structure and dynamics of the organization of the MAS.

Cassiopeia models agents as implementers of roles, which are, in turn, abstracted into different layers (see below) and identified in an *iterative*, *incremental* manner, following either a bottom-up approach (i.e. proceeding from the Domain-Dependent Role Layer), top-down approach (i.e. proceeding from the Organizational Role Layer) or a combination. This is compatible with OPF's “Iterative, Incremental, Parallel Life Cycle” model.

With respect to lifecycle phases, Cassiopeia proceeds from the definition of the system collective task to the design of MAS, thus being supported by the OPF Phases of “Initiation” and “Construction”.

The main modelling concepts in Cassiopeia are *role*, *agent*, *dependency*, and *group* where an agent’s roles are distinguished into three layers:

Domain-dependent roles: behaviours that individual agents perform

Relational roles: descriptions of how agents interact with each other, given the mutual dependencies of their domain-dependencies

Organizational roles: descriptions of how agents manage their interactions to dynamically organize themselves into groups

3. CASSIOPEIA TASKS, TECHNIQUES AND WORK PRODUCTS IN AND THEIR SUPPORT IN OPF.

In this section, we identify process component descriptions within the Cassiopeia documentation, captured here as instances of elements in the OPF metamodel. In particular, we seek Tasks, Techniques and Work Products. For each of these three elements, we analyze the Cassiopeia descriptions and then recast them into the OPEN Process Framework method engineering approach. This leads us to propose two new Tasks with two new subtasks and one new Work Product for addition to the OPF 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 OPF repository add to those already proposed to support agent-orientation in e.g. [11,12,13].

3.1 Tasks: Cassiopeia and the OPF

For each Cassiopeia task identified, we first describe it and then create a parallel OPF method chunk.

3.1.1 "Defining Domain-Dependent Roles"

Description: Roles are sets of behaviours that are put into operation by the agents to achieve the collective system task. In this task, after identifying the behaviours required for the system task and the set of roles encapsulating these behaviours, the designer should define agents by the set of roles they can play.

Support from OPF: The identification of system behaviours is addressed to some extent by tasks within the OPF Activity of "Requirement Engineering" and by Task: "Construct the Object Model". However, due to the different nature of "system requirements" and "system behaviours", the above OPF tasks should be extended to explicitly address the analysis/identification of system behaviours. We thus suggest introduction of a new task called "Identify system behaviours", which takes as inputs outputs from other OPF Requirement Engineering tasks such as "Elicit Requirements", "Analyze Requirements" and "Specify Requirements".

The identification and modelling of domain-dependent roles can be supported by Task "Identify agent's role" recommended by [14], together with OPF Task "Identify CIRTs" and a new Task: "Construct the Agent Model" to parallel the OO “Construct the Object Model”. The existing OPF Task "Map roles on to classes" can also be used to support the identification of agent classes in Cassiopeia.

TASK NAME: Identify system behaviours
Focus: Modelling of system functionality
Typical supportive techniques: Functional analysis techniques such as CRC modelling, Scenario development, Hierarchical task analysis, Responsibility identification, Service identification.
Explanation: Elementary activities required to fulfil the system’s goals and/or tasks need to be identified. These will later be performed by the system actors (in this case, agents) to achieve the collective goals and/or tasks.

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 their connectivity with other agents and objects.

3.1.2 "Defining Relational Roles"

Description: This Cassiopeia task analyses the organizational structure of an MAS based on the dependencies between domain-dependent roles. It includes two sub-tasks:

- Identify the dependencies between Domain-Dependent Roles, thereafter between agents.
- Determine relational roles of agents

Regarding the former sub-task, inter-role dependencies can be functional (i.e. when they are derived from behaviours implemented by domain-dependent roles), or relational (i.e. when they take place at the abstraction level of roles e.g. goal-based dependencies). Inter-role dependencies can then be naturally translated into dependencies between agents playing these roles. The modelling of inter-role/inter-agent dependencies is done by a Coupling Graph and Influence Graph (see Section 3.3 for more detail).

Regarding the latter sub-task, an agent involved in a dependency can play one of two relational roles: the role of an *influencing agent* or role of an *influenced agent*. (Note: Cassiopeia resorts to the abstract notion of “influence”: an influence relationship between an agent A and an agent B relies on an existing dependency between the domain-dependent role played by A and the domain-dependent role played by B). The names of the relational roles are determined by the dependency being considered, e.g. the *inhibition* dependency would give rise to the relational roles of *inhibitor* and *inhibited*. In this sub-task, the designer should also define:

- The influence signs, which can be roughly understood as interaction messages/commands sent by the influencing agent to the influenced agent. The designer should also take into account influence signs from sources other than agents, e.g. the environment.
- The relational behaviours that enable the agents to identify and handle the influence signs, i.e. how the influenced agent can choose among several influences to handle, which domain-dependent role the influenced agent should activate and in what fashion.

Support from OPF: The task of defining agents' relational roles (i.e. "influencing" role or "influenced" role), influence signs, and relational behaviours roughly corresponds to OPF's task "Construct the Agent Model". The specification of influence signs and relational behaviours, in particular, can be reasonably mapped to tasks "Determine agent interaction protocol" and "Determine agent communication protocol" of [14]. If the influence signs come from the environment, the OPF Task "Model the agent's environment" [14] can be used to identify these influence signs. The specification of inter-role/inter-agent dependencies (thereafter agents’ relational roles) can be supported by the OPF Task "Model dependencies for actors and goals" [11]. An alternative (though not one taken as yet) is to introduce a new task called "Model agent relational roles/dependencies" as a sub-task of "Construct the Agent Model".

3.1.3 "Defining Organizational Roles"

Description: This Cassiopeia task models the dynamics of the multi-agent organization in terms of the instantiation of potential groups of agents in the system. It consists of:

- specifying the *organizational roles* that enable the agents to manage agent groups, i.e. the roles of *group initiator* and *group participant*

- specifying the *organizational behaviours* of agents when playing these organizational roles, i.e. group formation behaviours, commitment behaviours, and dissolution behaviours

- defining the *influence signs* generated by these behaviours, e.g. commitment signs and dissolution signs.

Support from OPF: The dynamic self-organization of system components is not explicitly addressed by any existing OPF Task, although [14] offers the Task "Identify System Organization". This needs extension to includee the dynamic system organization issue. We thus introduce new subtasks, "Determine agents' organizational roles" and "Determine agents' organizational behaviours", for Task "Identify System Organization".

SUBTASK NAME: Determine agents' organizational roles
Focus: System dynamic modelling
Typical supportive techniques: Collaborations analysis, Control Architecture, Contract nets, Market Mechanisms
Explanation: At run-time, agents can dynamically organize/re-organize themselves. Roles of each agent within this dynamic organization should be predicted at the design time. For example, if agents need to form dynamic groups at run-time, organizational roles of “group initiator” and “group participant” need to be assigned to agents.

SUBTASK NAME: Determine agents' organizational behaviours
Focus: Agent functionality modelling
Typical supportive techniques: Collaborations analysis, Control Architecture, Contract nets, Market Mechanisms
Explanation: Activities to be performed, or rules to be adhered, by each agent when playing its organizational role need to be specified. These activities and rules define how the agent behaves and coordinates with other agents when the organization is formed, changed, or dissolved.

3.2 Techniques: Cassiopeia and the OPF

For each Cassiopeia technique identified, we first describe it and then create a parallel OPF method chunk.

3.2.1 For "Defining Domain-Dependent Roles"

Description: To identify system behaviours, Cassiopeia utilizes existing functional or OO analysis techniques. Given these behaviours, the designer should determine the appropriate level of abstraction so that the behaviours should achieve the proper functionality. For example, in the application of the soccer robot team (for which Cassiopeia was designed), potential system behaviours are "shoot", "place", "block", and "defend" - much more abstract than standard robot behaviours (e.g. turn left, right, accelerate).

For identification of domain-dependent roles, the designer should proceed in an iterative fashion, combining both the bottom-up approach (a behaviour focus) and a top-down approach (i.e. an organizational focus).

To identify agents based on roles, each agent can take on multiple or all of the identified roles, or only one role. In the former case, the agent can assign one domain-dependent role to act as the "active" role at a given time (while other roles are "idle"). This active role is determined by the agent’s relational role and organizational role at that point in time. The designer can also choose to design agents as either homogeneous (i.e. all agents are provided with the same set of domain-dependent roles), or heterogeneous (i.e. some agents are supplied with only a subset of these roles).

Support from OPF: For the identification of system behaviours, various analysis techniques from OPF can be useful, such as "CRC Modelling", "Scenario Development", "Hierarchical Task Analysis", "Responsibility Identification", and "Service Identification". The determination of an appropriate abstraction level for the system behaviours can be supported by OPF Technique "Abstraction Utilization". With regard to the specification of roles, OPF offers a "Role Modelling" technique, which only tackles the modelling of roles and does not address the task of role identification.

To identify agent classes from roles, OPF's Technique "Intelligent Agent Identification" is relevant, although this technique currently only targets the need for agents and agent modelling notation. Other OPF Techniques for OO class identification may be extended/adapted for agents, e.g. "Abstract Class Identification" and "Class Naming". The extension should take into account the major differences between OO classes and agent classes, e.g. agent classes are generally more coarse-grained than OO classes (thus, the "Granularity" Technique in OPF should be modified to support this).

3.2.2 For "Defining Relational Roles"

Description*:* For identification of inter-role dependencies and then inter-agent dependencies, the designer should consider various dependency types such as coordination, conditioning, simultaneous or sequential facilitation. Inconsistent dependencies should be removed, and when necessary, some dependencies can be ignored according to the available heuristics of the application domain. The designers should only retain inter-role/inter-agent dependencies that are relevant to the collective task achievement.

Cassiopeia provides no techniques for the specification of influence signs among agents. However, it does suggest that signs of influence produced by an “influencing” agent should correspond to the domain-dependent role it is playing. An “influenced” agent should be able to interpret these signs in so as to activate the appropriate domain-dependent role. In the exemplar soccer robot application, the potential influence signs are “help”, “shoot”, “place” and “position” messages.

Support from OPF: For the identification of inter-role/inter-agent dependencies (thereafter agents' relational roles), the OPF Technique "Collaboration Analysis" can be used. The specification of influence signs can be supported by the OPF Technique of "Interaction Modelling", while the determination of agents' relational behaviour can be supported [14] by Techniques such as "Deliberative Reasoning" and "Reactive Reasoning".

3.2.3 For "Defining Organizational Roles"

Description: Identification of organizational roles and behaviours of agents should be guided by the dependencies between domain-dependent roles and between agents. Specifically, an agent playing the role of “group initiator” should be the one involved in a dependency relationship, producing some influence signs to other agents. The initiator agent then evaluates its potential “teammates” (i.e. agents playing roles dependent on its own) to decide which are the most appropriate group members.

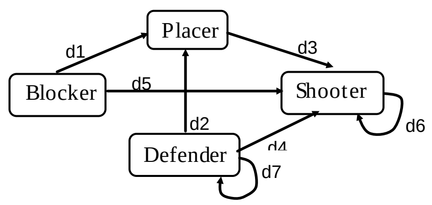
Regarding techniques for determining organizational behaviours (i.e. group formation, commitment and dissolution), Cassiopeia refers readers to other work such as Contract Net (already supported in the OPF[14]). Exemplar commitment behaviours are that the participant agents only activate particular domain-dependent roles or only respond to the initiator's influence signs. Group dissolution can occur when the initiator agent is satisfied or when a group can be replaced with a more efficient group.

Support from OPF: Determination of agents' organizational roles and behaviours Is supported by various Techniques found in the OPF repository of method chunks e.g. "Collaborations Analysis", "Control Architecture", "Contract Specification", "Contract Nets" "Commitment Management" [14].

3.3 Work Products: Cassiopeia and the OPF

The only two work products explicitly described in the initial version of Cassiopeia [8] are Coupling Graph and Influence Graph. The former shows dependencies between domain-dependent roles (Figure 1), while the latter is derived from the former to show dependencies between agents (Figure 2). Later versions [9], however, group these two into a single work product – the Coupling Graph – which is equivalent to the earlier Influence Graph [8].

The paths in the Coupling and Influence Graphs define the potential groupings of different domain-dependent roles (and hence agents), thus providing global representation of the organizational structure of the MAS.



d1: Blocking an opponent can help an agent to better place itself
d2: Defending can help oneself or another agent to better place itself
d3: Shooting depends on the position of oneself or opponent
d4: Defending may allow to catch the ball of the opponent
d5: Blocking can help oneself or another agent to shoot the ball

Figure 1. Coupling Graph of [8]

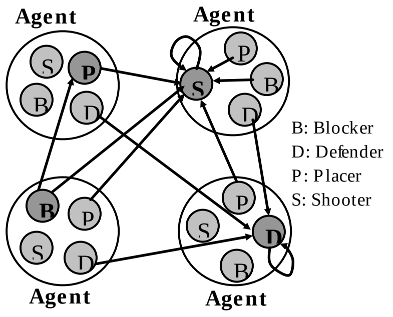
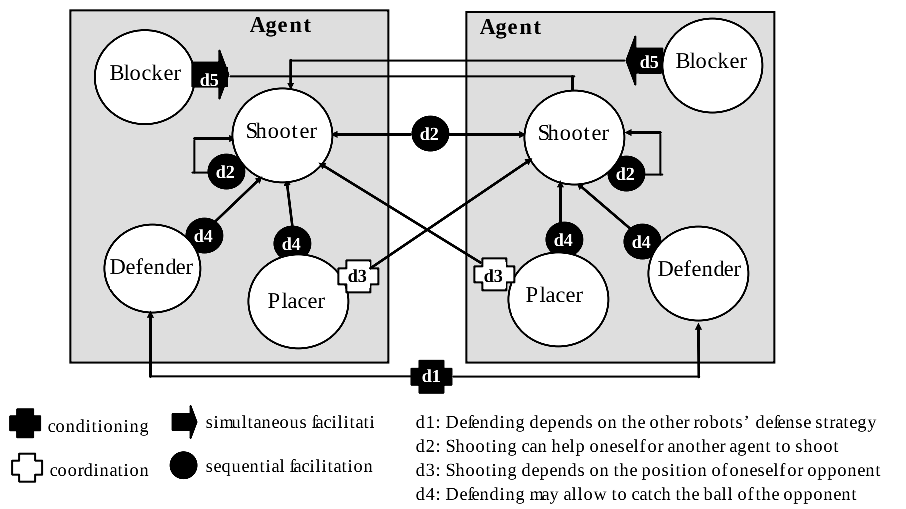


Figure 2. Influence Graph of [8]



d1: Defending depends on the other robots' defense strategy
d2: Shooting can help oneself or another agent to shoot
d3: Shooting depends on the position of oneself or opponent
d4: Defending may allow to catch the ball of the opponent

Figure 3. Coupling Graph of [9]

Although UML Collaboration Diagrams can be adapted/extended to cater for the Cassiopeia Coupling Graph and Influence Graph, these two models should be listed as new work products in OPF repository - or preferably included as a single model equated to Cassiopeia's later Coupling Graph as shown in Figure 3.

NAME: Coupling Graph
OPF CLASSIFICATION: Dynamic behavior diagrams
RELATIONSHIP TO EXISTING WORK PRODUCT: None
BRIEF DESCRIPTION: This diagram shows dependencies between agents and between roles encapsulated inside agents.

4. SUMMARY AND ACKNOWLEDGEMENTS

As part of an extensive research programme to combine the benefits of method engineering and to extend an existing object-oriented framework (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 two new Tasks, together with two new subtasks for a pre-existing Task plus one additional Work Product for inclusion in the OPF repository.

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

REFERENCES

1. Brinkkemper, S., 1996, Method engineering: engineering of information systems development methods and tools, *Inf. Software Technol.*, 38(4), 275-280.
2. Ter Hofstede, A.H.M. and Verhoef, T.F., 1997, On the feasibility of situational method engineering, *Information Systems*, 22, 401-422
3. 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
4. Ralyté, J. and Rolland, C., 2001, An assembly process model for method engineering, *Advanced Information Systems Engineering*), LNCS2068, Springer, Berlin, 267-283
5. Brinkkemper, S., Saeki, M. and Harmsen, F., 1998, Assembly techniques for method engineering. *Procs. CAISE 1998*, Springer Verlag, 381-400.
6. Henderson-Sellers, B., 2003, Method engineering for OO system development, *Comm. ACM*, 46(10), 73-78
7. Firesmith, D.G. and Henderson-Sellers, B., 2002, *The OPEN Process Framework. AN Introduction*, Addison-Wesley, Harlow, Herts, UK
8. Collinot, A. Drogoul, A. and Benhamou, P. 1996. Agent oriented design of a soccer robot team. *Procs. Second Intl. Conf. on Multi-Agent Systems (ICMAS'96)*
9. Collinot, A. and Drogoul, A. 1998. Using the Cassiopeia Method to Design a Soccer Robot Team. *Applied Articial Intelligence (AAI) Journal*, 12, 2-3, 127-147.
10. 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
11. Henderson-Sellers, B., Giorgini, P. and Bresciani, P., 2004, Enhancing Agent OPEN with concepts used in the Tropos methodology, *Procs. ESAW'0,* Springer (in press)
12. Henderson-Sellers, B. and Debenham, J., 2003, Towards OPEN methodological support for agent-oriented systems development, *Procs. First International Conference on Agent-Based Technologies and Systems*, University of Calgary, Canada, 14-24
13. Henderson-Sellers, B., Debenham, J. and Tran, N., 2004, Incorporating the elements of the MASE methodology into Agent OPEN, *Procs. ICEIS2004* (in press)

14. 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