INCORPORATING ELEMENTS FROM CAMLE IN THE OPEN REPOSITORY

C. Gonzalez-Perez, B. Henderson-Sellers, J. Debenham and G.C. Low, Q.-N.N. Tran
University of Technology, Sydney, Australia
University of New South Wales, Australia

Abstract: The CAMLE approach offers a methodological framework for the development of multi-agent systems. However, this approach does not provide full coverage of the needs often found in information systems development, lacking, for example, an appropriate capability for customization or links to infrastructural, non-engineering processes. By adopting a method engineering perspective, it is possible to integrate the best parts of CAMLE into the OPEN repository so organizations can create and own customized variants of CAMLE as necessary.

Key words: agent-oriented methodologies, CAMLE, OPEN, method engineering

1. INTRODUCTION

The CAMLE method [12] provides a fairly comprehensive language and environment for multi-agent software systems development. As with many other methods, it has been defined using natural language, which lacks the level of formalism needed to cater for extensions and customizations.

While individual methodologies may be useful for creating specific agent-oriented information systems, they have no flexibility as systems requirements change over time or as new concepts are understood and accepted. In contrast to these very rigid, single methodologies, the concept of situational method engineering (SME) provides a much more flexible environment in which to create a standard and useful methodological approach for systems design ([13], [10]). Here, we utilize one particular,

SME-focused methodological approach – that provided by the OPEN Process Framework [3], originally developed for creating object-oriented systems, but more recently expanded into "Agent OPEN" to include support for agent-oriented systems development [2, 6, 7]. As well as supporting SME, OPEN is formally defined using a metamodel. This means that individual method fragments are easily generated as instances of classes in the metamodel and, consequently, specific methods can be derived. In this paper, we show how the OPEN repository can be used to generate CAMLE. However, before this is possible, we need to evaluate the extent of existing support (in terms of repository-housed method fragments) in OPEN, particularly with respect to some of CAMLE's idiosyncratic features for multi-agent systems development.

In this paper, the major aspects of CAMLE are captured and integrated into OPEN. This is beneficial for both approaches, since CAMLE obtains a higher level of formalism (by being re-defined from a metamodel) and the OPEN repository is augmented with new contents. Section 2 briefly describes CAMLE and its main features, focussing on the major characteristics that differentiate it from other agent-oriented methods. Section 3 describes the OPEN Process Framework and its underpinning paradigm of method engineering. Section 4 provides a catalogue of new method fragments that are defined within the OPEN repository from CAMLE specifications. Finally, Section 5 presents our conclusions.

2. BRIEF DESCRIPTION OF CAMLE

CAMLE, as defined by its authors, is a caste-centric agent-oriented modelling language and environment [12]. It is caste-centric because *castes*, analogous to classes in object-orientation, are argued to provide the major modelling artefact over the lifecycle by providing a type system for agents. A significant difference is claimed between castes and classes: while objects are commonly thought of as statically classified (i.e. an object is created as a member of a class and that is a property for its whole lifetime), agents in CAMLE can join and leave castes as desired thus allowing dynamic reclassification.

CAMLE provides a graphical notation for caste models (similar to class models in OO methodologies), collaboration models and behaviour models. Caste diagrams are similar to conventional class diagrams in that they depict the types involved in the system as surrogates for the actual instances that will comprise it at run-time In CAMLE, these instances are agents and types are castes. In addition to inheritance and UML-style composition and

aggregation relationships (see [9], sec. 2.7.2.2), caste diagrams include notation for *congregation*, a kind of whole/part relationship in which the parts are detached from their type (i.e. caste) when the whole is destroyed; this is possible in CAMLE since dynamic classification of instances is fully supported. Similarly, notation for *migration* and *participation* relationships is available, representing cases of agents leaving their caste and joining a new one (migration) or agents that join a new caste without leaving their current ones (participation).

Collaboration diagrams depict the interactions between agents or castes. Unlike in UML collaboration diagrams ([9], sec. 3.65), interactions in CAMLE are produced by an agent observing the actions of other agents rather than by direct message sending or operation invocation. Therefore, arcs in CAMLE collaboration diagrams are labelled with an action name of the source agent rather than an operation name of the target. This suits well the autonomous and loose coupling nature of agents.

Behaviour models can be expressed in CAMLE using two kinds of diagrams. Scenario diagrams take the perspective of a given caste, describing a specific situation to which it must respond and what the response must be. A scenario diagram is necessary for each situation. Multiple scenario diagrams can be combined into a behaviour diagram, similar to a UML activity diagram ([9], sec. 3.84), which depicts the overall behaviour of a given caste.

From the process side, CAMLE defines three stages:

- the analysis and modelling of the existing information system
- the design of the new system as a modification of the existing one
- the implementation of the new system

CAMLE relies heavily on the fact that an information system already exists when a new project is started, so that the new system is designed as a modification to the current one. Although this situation is indeed common, the construction of systems from scratch also happens. CAMLE, however, seems to ignore this possibility.

At the same time, CAMLE defines a set of six activities that comprise the process of agent-oriented analysis and modelling, supposedly covering the two first stages in the list above. These activities produce caste models, collaboration models and behaviour models using an iterative and recursive approach. Agent castes that are too complex to be directly implemented are treated as complete systems and decomposed into component castes, applying the same activities to them recursively.

3. BRIEF DESCRIPTION OF METHOD ENGINEERING AND OPEN

The method engineering paradigm, introduced by the works of, for example, Kumar and Welke [8], Brinkkemper [1], Rolland and Prakash [10], and Saeki [11], advocates the definition of a method (or methodology) as a collection of *method fragments*, i.e. self-contained chunks that define work products, tasks, techniques or processes[1] independently of their potential context. Once a repository of such components has been constructed, a method engineer can select a sub-set and connect them together into a *situated method.* It is called "situated" because it is constructed purposefully for a particular means, usually a specific project or organization.

The OPEN Process Framework (see, e.g., [3-5]) adopts a process engineering perspective and defines a *framework* composed of a metamodel plus a repository of method fragments. The metamodel establishes what concepts and relationships can be utilized to define these method fragments. For example, the OPEN metamodel defines the concepts of Activity ("a major Work Unit that models a cohesive yet heterogeneous collection of Tasks that achieves a related set of goals", [4], p. 98) and Work Product ("any significant thing of value that is developed during an Endeavour", [4], p. 65), and states that each activity produces zero or more work products ([4], fig. G.4). The OPEN repository is populated with method fragments that are instances of the concepts defined in the metamodel; following our example, the OPEN repository could contain an activity named "Requirements Engineering" (an instance of Activity) plus a work product named "System Requirements Specification" (an instance of WorkProduct), plus a link between them (an instance of the association between Activity and WorkProduct).

The OPEN metamodel is often seen as something fixed and given to method engineers "as is". Having an unchanging background on which everything else is based is useful for the common understanding and compatibility between methodologies. Each method engineer, however, is free to utilize the method fragments already pre-defined in the OPEN repository or create new ones if necessary. In fact, the addition of new method fragments to the repository comprises OPEN's major mechanism for extension and customization. This paper studies the CAMLE method and creates new OPEN repository method fragments based on CAMLE's specification, thus incorporating the process and work product definitions of the latter. Once this is done, CAMLE exists embedded in the OPEN

[1] We are using these terms here in their conventional sense. No technical meaning is attached.

repository as a collection of method fragments, and CAMLE itself or any variant of it, even those involving method fragments other than CAMLE's, can be generated.

4. INTEGRATING CAMLE ELEMENTS INTO OPEN

This section explores different aspects of CAMLE from the perspective of the OPEN metamodel. For those CAMLE aspects that can be satisfactorily represented by an already-existing method fragment, this method fragment is identified. For those aspects of CAMLE that cannot be modelled by any method fragment in the OPEN repository, a new method fragment is introduced and fully defined.

4.1 Lifecycle

An OPEN lifecycle is the span of time associated with the development of a given software system. In this respect, CAMLE uses an iterative and recursive lifecycle that fits well into OPEN's *Iterative, Incremental, Parallel Lifecycle*. No new method fragments need to be introduced.

4.2 Phases and Activities

An OPEN phase is a large-grained span of time within a lifecycle that works at a given level of abstraction. An OPEN activity is a large-grained unit of work that specifies what to do in order to accomplish some results. Activities state *what* to do, while phases set the temporal frame stating *when* to do it.

CAMLE does not make a difference between what and when to do things. On the contrary, it defines three "stages" that involve work description and temporal issues altogether. First of all, the existing information system is analysed and modelled, which is likely to comprise an appropriate mix of requirements engineering and legacy mining. OPEN's activities *Requirements Engineering* plus probably *Component Selection* and *Environment Engineering*, assembled into an *Initiation* phase, are appropriate to model this.

Secondly, the new system is designed as a modification of the existing one. This is basically a design job that can be satisfactorily represented by OPEN's activity *Design*. Finally, the new system is implemented, which can be modelled by OPEN's activities *Implementation* and *Integration*. All these design and implementation activities can be packaged into a *Construction* phase.

CAMLE always assumes that an existing system is present and that the new system is constructed as a modification from it. OPEN does not impose this perspective on the phases of the lifecycle, or any other for that matter. In any case, the three "stages" defined by CAMLE can be easily mapped to already existing OPEN activities and phases, and therefore no new method fragments need to be introduced.

4.3 Tasks

An OPEN task is a small-grained, atomic unit of work that specifies what must be done in order to achieve some stated result. CAMLE defines six "activities", all within the second "stage", namely system design (see Section 4.2). No "activities" are described for the other two "stages". CAMLE "activities" semantically correspond to OPEN's tasks.

First, agents and their roles in the system are identified according to their functionality and responsibilities, and then grouped into castes. This produces a caste model. Agent OPEN's *Analyse use requirements*, *Identify CIRTs* and, most importantly, *Model agents' roles* and *Construct the agent model* are suitable tasks to model this. CIRT stands for "class, instance, role or type" and, in the context of CAMLE, the concept of CIRT can be extended to castes as well.

Then, inheritance and whole/part relationships between castes are modelled, augmenting the caste model with new information. Agent OPEN's *Construct the agent model*, again, can be used to model this.

The third "activity" involves identifying the communication links between castes in terms of how agents influence each other, producing a first version of the collaboration model. Agent OPEN's *Evaluate the design* (since the already existing caste model must be evaluated and possibly modified) and *Construct the agent model* can be used to model this.

Then, visible actions and state variables of each caste are identified and associated with communication links in the collaboration model. OPEN does not include any task appropriate to representing the identification of actions, so a new one must be introduced. Agent OPEN does, however, include the task *Construct the agent model*, which is appropriate to model the identification of state variables.

The newly proposed OPEN Task (to support CAMLE) is defined as follows:

OPEN Task

Task Name:	*Determine Caste Actions*
Relationships:	Semantically close to *Identify CIRTs* and *Construct the object model.*
Focus:	Dynamic modelling
Typical supportive	Techniques: *Responsibility identification*, *Responsibility-driven design*, *Delegation analysis*, *Event modelling*, *Interaction modelling*
Explanation:	This task defines the visible actions that a given agent caste is capable of performing.

Then, scenarios in the operation of the system as a whole are identified. OPEN's *Analyse user requirements* and *Design user interface* can be used to model this. Subtask Use case modelling deals predominantly with use cases using techniques such as *Hierarchical task analysis* and *Scenario development.*

Finally, each caste's response to each of the identified system scenarios is analysed and described, producing a behaviour model for each caste. Design-focussed tasks in OPEN, useful here, include *Capture the design*, *Document the design* and *Refactor*; with many techniques such as *Collaborations analysis*, *Relationship modelling* and *Robustness analysis.*

4.4 Work Products

An OPEN work product is an artefact of interest for the performed work units. Work products usually comprise models and documents. CAMLE defines several kinds of diagrams, which can be modelled as OPEN work products.

A caste diagram in CAMLE is similar to a conventional class diagram, but it incorporates new constructs such as congregation, migration and participation relationships. In addition, the CAMLE literature does not mention the usage of UML-style associations ([9], sec. 3.41) in caste diagrams, which is probably an additional difference. Therefore, a new method fragment needs to be introduced.

OPEN Work Product

Name:	*Caste Diagram*
OPEN classification:	Static Architecture set of work products
Brief description:	A caste diagram depicts the castes in a system, together with the inheritance and whole/part relationships among them. Whole/part relationships include the conventional composition and aggregation relationships plus the newly proposed relationships of congregation, migration and participation [12].

A collaboration diagram in CAMLE is similar to a UML collaboration diagram, but incorporates the major difference of having the arcs labelled with a sender's action rather than a receiver's operation. This is a large enough distinction as to require the introduction of a new method fragment.

OPEN Work Product

Name: *Caste Collaboration Diagram*
OPEN classification: Dynamic Behaviour set of work products
Brief description: A caste collaboration diagram depicts the interactions between agent castes by showing what actions of what castes are observed (and reacted to) by other castes. Arcs in the diagram are labelled with the source caste's action name.

Behaviour models in CAMLE describe how a specific caste behaves from its own perspective. In particular, scenario diagrams show how agents of a given caste act when in a given scenario. Although scenario diagrams are similar to UML statechart diagrams ([9], sec. 3.74), they incorporate additional notation for predicates, state assertions and logic connectives. A new method fragment is necessary to reflect this.

OPEN Work Product

Name: *CAMLE Scenario Diagram*
OPEN classification: Dynamic Behaviour set of work products
Brief description: A CAMLE scenario diagram depicts the sequence of actions and state assertions that agents of a given caste go through when in a given scenario. Single and repetitive actions, as well as simple and continuous state assertions can be depicted, using logical connectives (and, or, not) if necessary.

Behaviour diagrams combine all the scenario diagrams of a given caste into a single diagram showing its overall behaviour and hiding the details of each single scenario. A new method fragment is introduced here.

OPEN Work Product

Name: *CAMLE Behaviour Diagram*
OPEN classification: Dynamic Behaviour set of work products
Brief description: A CAMLE-derived behaviour diagram depicts the interrelations between scenarios, preconditions and actions of a given agent caste. All the notation available for scenario diagrams is available, as well as a conflux notation that can be used to represent a logical conjunction of scenarios and/or preconditions.

4.5 Other Method fragments

CAMLE does not provide any information on appropriate techniques, necessary roles or other kinds of method fragments. Therefore we must assume that the repository of OPEN method fragments is already rich enough for the implementation of CAMLE.

5. CONCLUSIONS

We have shown in this paper how the major elements of CAMLE can be re-defined formally in terms of the OPEN metamodel, and how the OPEN repository is augmented with method fragments made out of them. This results in an increased level of formality for CAMLE, which enables it to be extended, customized and integrated with other method fragments already present in the OPEN repository. It also results in a benefit for OPEN, since it gains additional content for its repository that enhances its capability to tackle agent-oriented systems development.

ACKNOWLEDGEMENTS

We wish to acknowledge financial support from the University of Technology, Sydney under their Research Excellence Grants Scheme and from the Australian Research Council. This is Contribution number 04/22 of the Centre for Object Technology Applications and Research.

REFERENCES

1. Brinkkemper, S., 1996. *Method Engineering: Engineering of Information Systems Development Methods and Tools.* Information and Software Technology. **38**(4): p. 275-280.
2. Debenham, J. and B. Henderson-Sellers, 2003. *Designing Agent-Based Process Systems - Extending the OPEN Process ~~Framework, in Intelligent Agent~~ Software Engineering*, V. Plekhanova (ed.). Idea Group. p. 160-190.
3. Firesmith, D.G., 2004. *Firesmith OPEN Process Framework (OPF) Website* (web site). Accessed on 27th April 2004. http://www.donald-firesmith.com/
4. Firesmith, D.G. and B. Henderson-Sellers, 2002. *The OPEN Process Framework.* The OPEN Series. London: Addison-Wesley.
5. Graham, I., B. Henderson-Sellers, and H. Younessi, 1997. *The OPEN Process Specification.* The OPEN Series. Harlow (Essex), UK: Addison-Wesley Longman.
6. Henderson-Sellers, B., J. Debenham, and Q.-N.N. Tran, 2004. *Adding Agent-Oriented Concepts derived from Gaia to Agent OPEN.* In *Procs. CAiSE 2004.* Springer-Verlag: Berlin, Germany.

7. Henderson-Sellers, B., J. Debenham, and Q.-N.N. Tran, 2004. *Incorporating the Elements of the MASE Methodology into Agent OPEN.* In *Procs. of the 6th International Conference on Enterprise Information Systems 2004.* 4.
8. Kumar, K. and R.J. Welke, 1992. *Methodology Engineering: a Proposal for Situation-Specific Methodology Construction*, in *Challenges and Strategies for Research in Systems Development*, W.W. Cotterman and J.A. Senn (eds.). John Wiley & Sons: Chichester, UK. p. 257-269.
9. OMG, 2001. *Unified Modelling Language Specification.* formal/01-09-68 through 80 (13 documents). Object Management Group.
10. Rolland, C. and N. Prakash, 1996. *A Proposal for Context-Specific Method Engineering.* In *Procs. IFIP WG8 International Conference on Method Engineering.* Atlanta, GA.
11. Saeki, M., 2003. *CAME: the First Step to Automated Software Engineering.* In *Procs. OOPSLA 2003 Workshop on Process Engineering for Object-Oriented and Component-Based Development.* Anaheim, CA, 26-30 October 2033. COTAR: Sydney.
12. Shan, L. and H. Zhu, 2004. *CAMLE: A Caste-Centric Agent-Oriented Modeling Language and Environment.* In *Third International Workshop on Software Engineering for Large-Scale Multi-Agent Systems.* Edinburgh, 24-25 May 2004. [in press]. Springer-Verlag.
13. ter Hofstede, A.H.M. and T.F. Verhoef, 1997. *On the Feasibility of Situational Method Engineering.* Information Systems. **22**(6/7): p. 401-422.