Review Article

Towards the Consolidation of a Diagramming Suite for Agent-Oriented Modelling Languages

Brian Henderson-Sellers

University of Technology, Sydney, Broadway, NSW 2007, Australia

Correspondence should be addressed to Brian Henderson-Sellers; brian.henderson-sellers@uts.edu.au

Received 21 October 2012; Accepted 7 November 2012

Academic Editors: X. He, S. Sutton, and M. Viroli

Copyright © 2013 Brian Henderson-Sellers. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Whilst several agent-oriented modelling languages have been developed by independent research groups, it is now appropriate to consider a consolidation of these various approaches. There are arguably three things that need consolidation and future standardization: individual symbols, the underpinning metamodel, and the diagram types. Here we address only the third issue by extending an earlier analysis that resulted in recommendations for various diagram types for the modelling of a multiagent system (MAS). Here, we take each of these previously recommended diagram types and see how each is realized in a wide variety (over 20) of current agent-oriented software engineering (AOSE) methodologies. We also take the opportunity to express, as exemplars, some of these diagram types using the recently published FAML notation.

1. Introduction

Any software development benefits from the use of a methodology. Part of such a methodological approach is a means to depict interim work products, typically documented using a graphical notation (a.k.a. concrete syntax). Symbols are used to represent single concepts, defined in an appropriate modelling language (ML), itself typically represented, at least in part, by a metamodel (e.g., [1]). These symbols can then be grouped in heterogeneous yet semantically related ways. A coherent model, thus depicted, is often said to be of a specific diagram type. In other words, a diagram type refers to a collection of classes in the metamodel, that is, it defines which metaclasses can be appropriately instantiated for this particular scope and focus.

For agent-oriented software engineering (AOSE), such modelling languages (and their notations and recommended diagram types) are in their infancy. A large number of AOSE methodological approaches exist, all with their own notational elements. As part of a community goal of standardizing agent-oriented modelling languages, collaborative notations have been proposed (e.g., [2]), as well as mergers at the conceptual level (e.g., [3, 4]), the latter of these being complemented more recently by a concrete syntax [5].

Notations need to have a high degree of usability, which can often be accomplished based on semiotic principles (e.g., [6–8]). Information is needed not only about individual agents and interagent communications, but also on the context of the environment in which they are situated. Current practice in many methodological approaches is to utilize standard object-oriented diagramming techniques, typically using UML [9–11] as a notation, whenever possible, although there are many concepts in AOSE not so representable. For example, Garcia et al. [12] comment on the need to include specific agenthood properties, including interaction/communication, autonomy, and adaptation with possible additional properties of learning, mobility, collaboration, and roles. A similar list, yet with a BDI (BDI = beliefs, desires, and intentions (e.g., [13, 14])) slant, is given by Sturm and Shehory [15, 16] as agent, belief, desire, intention, message, norm, organization, protocol, role, society, and task. Taveter and Wagner [17] identify the most important concepts as including agents, events, actions, communication, and messages, underpinning these in terms of ontological theory (e.g., endurants and perdurants). Bertolini et al. [18] focus primarily on the Goal Diagram and the Actor Diagram in their presentation of TAOM4E—an Eclipse-based tool to
support the Tropos methodology and based solidly on a metamodel.

Beydoun et al. [4] present a generic metamodel which itself contains four connected perspectives. In this case, the discrimination is between organization as compared to agent level and between design time and run time. However, they do not explicitly link these to diagram types, although there is in fact a weak relationship.

Diagram types are often divided into two loosely defined groups: static or structural diagrams and dynamic (a.k.a. behavioural) diagrams (e.g., [69, 70]) — a grouping that will also be utilized here. The former depict aspects that might be termed architectural, typified by variants of an OO class diagram; the latter depict some forms of functionality and time-dependent actions.

Torres da Silva et al. [71] have presented MAS-ML as a metamodel-based modelling language for agent-oriented software engineering. As well as introducing new agent-focussed concepts, as discussed below, they also recommend a suite of diagram types—three static and two dynamic:

(i) Extended UML Class Diagram,
(ii) Organization Diagram,
(iii) Role Diagram,
(iv) Extended UML Sequence Diagram,
(v) Extended UML Activity Diagram.

In contrast to the approach taken in the ML proposed by Beydoun et al. [4] that focusses first on a viewpoint and later on the detailed concepts, Torres da Silva et al. [71] propose not viewpoints but specific diagram types, although they neglect to give a clear problem statement for which these diagram types are the proposed solution. In other words, whilst useful, they are at the diagram level rather than the viewpoint level as advocated in Henderson-Sellers [19]. We will therefore comment on each of these diagrams in the appropriate place in Sections 5 and 6.

In summary, we aim here to make a contribution towards future standardization of agent-oriented modelling languages—focussing here on diagram suites. Section 2 outlines the approach taken in determining an appropriate framework, which we then use to analyze over 20 contemporary agent-oriented methodologies in terms of the kinds of diagrams that they support and recommend. Section 3 discusses notational aspects, introducing the FAML notation [5] that we use in later examples in comparison with the notations used by these individual AOSE methodologies. Following an overview of diagram types in Section 4, in the next two sections, we describe in detail static diagram types (Section 5) and then dynamic diagram types (Section 6). In each of these two sections, we categorize diagrams using the several views derived in Section 2. Section 7 provides a final discussion and indicates some other related work not otherwise cited followed by a brief conclusion section (Section 8) including some ideas for future research. From this detailed comparison, we aim to draw out commonalities and variations in the suite of diagram types utilized across all extant agent-modelling languages as a precursor to future international standardization.

<table>
<thead>
<tr>
<th>Static diagram types</th>
<th>Dynamic diagram types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment description</td>
<td>Agent goal-based use case</td>
</tr>
<tr>
<td>Environmental connectivity</td>
<td>Use case map</td>
</tr>
<tr>
<td>External organization structure chart</td>
<td>Conversation a.k.a. interaction</td>
</tr>
<tr>
<td>Architecture</td>
<td>Protocol (a kind of conversation)</td>
</tr>
<tr>
<td>Agent society</td>
<td>Workflow</td>
</tr>
<tr>
<td>Agent role</td>
<td>Agent state</td>
</tr>
<tr>
<td>Role dependency</td>
<td>Task specification</td>
</tr>
<tr>
<td>Agent internals</td>
<td>Task state</td>
</tr>
<tr>
<td>Agent overview</td>
<td></td>
</tr>
<tr>
<td>Goal decomposition</td>
<td></td>
</tr>
<tr>
<td>Ontology</td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td></td>
</tr>
<tr>
<td>Capability</td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td></td>
</tr>
<tr>
<td>Task decomposition</td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
</tr>
<tr>
<td>UI design</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Static textual diagram types</th>
<th>Dynamic textual diagram types</th>
</tr>
</thead>
<tbody>
<tr>
<td>System requirements</td>
<td>Goal-based use case template</td>
</tr>
<tr>
<td>Role definition template</td>
<td>Contractual template</td>
</tr>
<tr>
<td>Agent descriptors</td>
<td>Event descriptors</td>
</tr>
<tr>
<td>CRC cards</td>
<td>Data descriptors</td>
</tr>
<tr>
<td>Plan descriptor</td>
<td>Plan descriptors</td>
</tr>
<tr>
<td>Capability descriptors</td>
<td>Task template</td>
</tr>
<tr>
<td>Service diagram</td>
<td>Protocol descriptors</td>
</tr>
<tr>
<td>Task template</td>
<td>Message descriptors</td>
</tr>
<tr>
<td>Percept descriptor</td>
<td>Action descriptor</td>
</tr>
<tr>
<td>Process descriptor</td>
<td></td>
</tr>
</tbody>
</table>

2. Research Approach

As detailed in Henderson-Sellers [19], in order to analyze the various options for a suite of AOSE relevant diagrams, the first step was to identify static versus dynamic diagram types (Tables 1 and 2) and then to group these in terms of their relevance to a number of views or viewpoints as previously discussed in the AOSE literature (e.g., [20, 48]). Seven such views were identified (Table 3), and, for each, both static and dynamic diagram types were identified (Table 4). (Details of the several iterations needed to derive Table 4 are to be found by Henderson-Sellers [19] and are not replicated here.) Finally, the atomic elements identified for each of these diagram types are listed in Table 5. However, this list is not absolute in that different methodologies offer different interpretations and consequently use different atomic elements on
Table 3: Seven views recommended in the analysis of Henderson-Sellers [19]. Note that the original analysis was based on the AOSE literature which essentially eschews aspects of user interface. To these seven, an eighth one, UI, needs to be added (reprinted from [19], copyright 2010, with permission from IOS Press).

<table>
<thead>
<tr>
<th>View name</th>
<th>Focus of view</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>External context, including system requirements</td>
</tr>
<tr>
<td>Architecture</td>
<td>High level structure of system independent of agent technology</td>
</tr>
<tr>
<td>Agent societies</td>
<td>Structure of agents into groups together with interactions and information exchange, typically within the group</td>
</tr>
<tr>
<td>Agent workflow</td>
<td>Workflows</td>
</tr>
<tr>
<td>Agent knowledge</td>
<td>Roles of individual agents, their responsibilities, and purpose</td>
</tr>
<tr>
<td>Agent services</td>
<td>Services offered, tasks to be undertaken, goals to be attained, and detailed capabilities. Applied to a small number of interacting agents</td>
</tr>
<tr>
<td>Deployment</td>
<td>Interface with run-time platform</td>
</tr>
</tbody>
</table>

Table 4: Two dimensional matrix for views versus static/dynamic aspects for various AOSE diagram types (modified and reprinted from [19], copyright 2010, with permission from IOS Press).

<table>
<thead>
<tr>
<th>View</th>
<th>Static diagram types</th>
<th>Dynamic diagram types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>Environment description; environmental connectivity; system requirements; use case</td>
<td>N/A</td>
</tr>
<tr>
<td>Architecture</td>
<td>Agent societies/organization</td>
<td>N/A</td>
</tr>
<tr>
<td>Agent societies</td>
<td>Agent society details; agent role</td>
<td>Conversation (including interaction and protocol); task</td>
</tr>
<tr>
<td>Agent workflow</td>
<td>N/A</td>
<td>Workflow</td>
</tr>
<tr>
<td>Agent knowledge</td>
<td>Goal; agent type; agent role; plan; ontology</td>
<td>Goal; agent state; capability</td>
</tr>
<tr>
<td>Agent services</td>
<td>Agent society details; agent type; goal; ontology</td>
<td>Goal; task; capability</td>
</tr>
<tr>
<td>Deployment</td>
<td>Allocation to run-time platform</td>
<td>N/A</td>
</tr>
<tr>
<td>UI</td>
<td>User interface design</td>
<td>States and transitions related to interface</td>
</tr>
</tbody>
</table>

Any one named diagram—for example, Padgham et al. [2] note that in the Prometheus methodology an Agent Society model shows actions and percepts but would not use an Ontology diagram, whereas users of the PASSI methodology would use a separate Ontology diagram.

As the knowledge of AOSE increases, the diagram suite suggested in Table 4 and the details of Table 5 will almost certainly require further changes—this paper offers further comments based on further investigation of the extant literature.

An initial assessment [19] resulted in some suggested recommendations for each diagram type in Table 4. Here, we commence with those recommendations and evaluate how each particular diagram type is utilized in methodologies not previously discussed. With the recent advent of a proposed notational standard for FAML [5], we take the opportunity of including an evaluation of how the symbols in this modelling language (summarized in Figure 2) can be useful. In cases where problems are identifiable, this could lead to improvements to be proposed to the FAML notation itself.

3. Notations

Notations (a.k.a. concrete syntax) currently utilized for agent-oriented methodologies are typically individualistic. However, there are efforts under way to systematize these. Two proposed notations, AML [72, 73] and AUML [74, 75], are essentially extensions of an object-oriented modelling language—whether this is appropriate is discussed in, for example, Torres da Silva and de Lucena [76], Choren and Lucena [77], and Beydoun et al. [4].

In AML, UML class diagrams are used with subtypes of Ontology Diagrams, Society Diagrams, Behavior Decomposition Diagrams, and Mental Diagrams (with a further subtype of Goal-Based Requirements Diagram). Composite Structure Diagrams (from UML) can be either Entity or Service Diagrams in AML; UML sequence diagrams are used as Protocol Sequence Diagrams with a subtype of Service Protocol Sequence Diagram. Finally, UML communication diagrams are realized as Protocol Communication Diagrams, a subtype of which is the Service Protocol Communication Diagram.

These UML-based notations are not readily related to the seven views identified by Henderson-Sellers [19] (see Table 3), although they do discuss static versus dynamic aspects of each diagram (Table 1).

Secondly, a number of methodologies use as their main notation that of $i^*$ [78] (later mapped in the agent-oriented context to UML by Mylopoulos et al. [79]). Designed for requirements engineering, $i^*$’s usage in AOSE has been primarily in the requirements and architectural design stages of Tropos (in later stages Tropos uses AUML/UML diagrams)
Table 5: Atomic elements and diagram types.

<table>
<thead>
<tr>
<th>Diagram type</th>
<th>Atomic elements to be displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Static diagram types</td>
<td></td>
</tr>
<tr>
<td>Environment description</td>
<td>Entities represented by classes; relationships between the modelled entities</td>
</tr>
<tr>
<td>Environmental connectivity</td>
<td>Agents/MASSs, internal and external resources, relationships across the MAS/environment interface</td>
</tr>
<tr>
<td>External organization structure chart</td>
<td>Organizational units in the real-life business</td>
</tr>
<tr>
<td>Architecture</td>
<td>Technology-independent large-scale structure</td>
</tr>
<tr>
<td>Agent society</td>
<td>Agents inside the MAS, how they associate with each other</td>
</tr>
<tr>
<td>Agent role</td>
<td>Links between the agents and the roles they play</td>
</tr>
<tr>
<td>Role dependency</td>
<td>Hierarchical structure of many roles</td>
</tr>
<tr>
<td>Agent internals</td>
<td>Constituent elements in an individual agent or role</td>
</tr>
<tr>
<td>Agent overview</td>
<td>High level view of an agent</td>
</tr>
<tr>
<td>Goal decomposition</td>
<td>Goals, subgoals</td>
</tr>
<tr>
<td>Ontology</td>
<td>The underpinning semantic structure</td>
</tr>
<tr>
<td>Plan</td>
<td>The (process) steps needed to effect a task and accomplish a goal</td>
</tr>
<tr>
<td>Capability</td>
<td>The ability or responsibility of an agent</td>
</tr>
<tr>
<td>Service</td>
<td>Functionality offered by the agent</td>
</tr>
<tr>
<td>Task decomposition</td>
<td>Tasks, subtasks</td>
</tr>
<tr>
<td>Deployment</td>
<td>Allocation of MAS elements to nodes of the run-time platform</td>
</tr>
<tr>
<td>UI design</td>
<td>TBD (the topic of proposed future research). (See brief discussion in Sections 5.7 and 6.5 on the relevant, non-AOSE UI literature)</td>
</tr>
<tr>
<td>(2) Dynamic diagram types</td>
<td></td>
</tr>
<tr>
<td>Agent goal-based use case</td>
<td>Functionality offered by the MAS</td>
</tr>
<tr>
<td>Use case map</td>
<td>Threads across many agents to realize a use case</td>
</tr>
<tr>
<td>Conversation</td>
<td>Dynamic interaction details</td>
</tr>
<tr>
<td>Protocol</td>
<td>Rules associated with interactions</td>
</tr>
<tr>
<td>Workflow</td>
<td>Large-scale processes relating to problem solving (in the real world)</td>
</tr>
<tr>
<td>Agent state</td>
<td>Attribute values determining the current state of an agent</td>
</tr>
<tr>
<td>Task specification</td>
<td>Definitions of tasks needed to accomplish a specific goal</td>
</tr>
<tr>
<td>Task state</td>
<td>The current state of a task, in terms of how far through the task enactment</td>
</tr>
</tbody>
</table>

because that agent-oriented methodology uses requirements engineering concepts throughout the development process. However, more recently this notation has been more widely evaluated. For example, Lapouchian and Lespérance [80] map between $i^*$ and CASL (Cognitive Agents Specification Language [81]) representing agents’ goals and knowledge as mental states; Franch [82] assesses the predictability of $i^*$ models; Estrada et al. [83] undertake an empirical evaluation of $i^*$ using industrial case studies and conclude that extensions and modifications are needed for $i^*$ to address its lack of modularization.

Although most methodologists devise their own notation, there has been over the last few years a groundswell of opinion that notations (and metamodels) should be applicable to more than just a single methodological approach. In that spirit, Padgham et al. [2] suggest a notation based on a merger between the notations that are part of O-MaSE, Tropos, Prometheus, and PASSI. (Sources/citations for the various AOSE methodologies are found in Table 7). Although a huge step forward in the future creation of a widely acceptable standard AOML, Henderson-Sellers et al. [5] offered some areas for improvement, based on semiotic considerations. Using that experience (of Padgham et al. [2]), they then offered a notation that has a stronger semiotic basis whilst retaining ideas from Padgham et al. [2] when appropriate. This notation has elements that are conformant to the FAML metamodel of Beydoun et al. [4].

In their definition of a modelling language, which contains more detail than we seek at present, Beydoun et al. [4] split their metamodel diagrams into four parts, which correspond interestingly with the viewpoints discussed in Henderson-Sellers [19] and outlined above. Beydoun et al. [4] discriminate between internal versus external (to an agent) and design versus runtime perspectives. Their System-level diagram corresponds to the Organization view of Table 3 together with some aspects of the Knowledge view (specifically in terms of role modelling) and their Environment level diagram to the Environment view. Their agent-definition metamodel fragment depicts specifications for agent types, messages, and plans, inter alia, and would therefore seem to have a reasonable correlation with the Services view in Table 3, whereas the agent-level (runtime) portion of the
metamodel goes somewhat beyond the views of Table 3, since it describes metamodelling support for the runtime “Actions” of individual agents, moving on from plan descriptors, for example, to plan enactment. Run-time concepts can thus be linked to some of the dynamic diagram types discussed by other authors (and one of the two discriminators used in this survey). An important distinction is made between agent types (the equivalent to OO classes in a class diagram) and (runtime) agents, which are individuals (equivalent to objects in an OO environment) (see also [59, page 93]). This distinction was made after surveying the literature wherein agent types are often (mis)labelled agents.

The initial studies for the derivation of FAML’s metamodel and notation were confined to what might be called “basics” (Figure 1), in that they did not take into account security, mobility, or trust. These are to be regarded as FAML Extensions, the detailed derivation of which is yet to be undertaken.

The set of symbols proposed for the FAML Basics (Figure 1) by Henderson-Sellers et al. [5] have since been slightly modified as a result of questions and discussions at the conference presentation. Figure 2 shows this final set, which we evaluate further in this paper. The principles behind the choice of symbol include ease of drawing, that “families” of symbols should have the same shape and colour (Table 6) and that colour should be an enhancer and not a determinant; that is, the shapes should be understandable in black and white as well as colour. These, and other principles, accord well with the semiotic discussion and principles of Constantine and Henderson-Sellers [6, 84] and Moody [7].

Symbols for agents and roles utilize the role “mask” and its variations. Process-style symbols are similar to those in ISO/IEC [21], topologically similar and green in colour. Events and resources, whilst being a little difficult to defend as a “family”, have, nevertheless, similar shapes and colours. Goals, on the other hand, are linked to beliefs as part of the mental state of agents. They use a familiar representation using Yu’s [78] notation, as used in agent methodologies like Tropos and Secure Tropos [85]. When used, the fill colour is brown. Ontology, service, and capability are grouped together because both Service and Ontology are linked to Role in FAML. Finally, both scenarios and actors can be linked by their common usage in use case style diagrams. For these, we simply adopt the symbols proposed by Padgham et al. [2].

Agent interactions utilize various variants of an arrowhead (Figure 3). Two alternatives (for MessageIn and MessageOut) were also proposed, but discussants at the EMM-SAD conference in June 2012 at which these ideas were first presented were undecided whether the symbols in Figure 3 or in Figure 4 were preferable. Here, we use those of Figure 3.

---

**Figure 1:** Organizational structure of FAML into basic elements and extensional elements.

**Table 6: Initially proposed families and their members.**

<table>
<thead>
<tr>
<th>Family</th>
<th>Members</th>
<th>Shape</th>
<th>Colour (optional)</th>
<th>Source and/or influence for notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agents and roles</td>
<td>Agent, role, group, position, organization</td>
<td>Circle atop mask or rectangle</td>
<td>Yellow</td>
<td>INGENIAS [20]</td>
</tr>
<tr>
<td>Tasks and plans</td>
<td>Action specification, FAML task, plan specification</td>
<td>Curvilinear</td>
<td>Green</td>
<td>ISO/IEC [21]</td>
</tr>
<tr>
<td>Events and resources</td>
<td>Event, resource</td>
<td>Triangular</td>
<td>Blue/green</td>
<td></td>
</tr>
<tr>
<td>Goals</td>
<td>Hard goal, soft goal, belief</td>
<td>Complex curvilinear</td>
<td>Brown</td>
<td></td>
</tr>
<tr>
<td>Ontology</td>
<td>Ontology, service, capability</td>
<td>Polygonal</td>
<td>Dark blue</td>
<td></td>
</tr>
<tr>
<td>Use cases</td>
<td>Scenario, actor</td>
<td>Double oval, stick figure</td>
<td>None</td>
<td>Padgham et al. [2]</td>
</tr>
<tr>
<td>Messages</td>
<td>Conversation, message in, message out</td>
<td>Arrowheads</td>
<td>B/W</td>
<td>Padgham et al. [2]</td>
</tr>
</tbody>
</table>

Strictly agent types and role types (design time concepts) rather than their run-time equivalents of individual agents and individual roles.
4. Diagram Types Used in Current AOSE Methodologies

Henderson-Sellers [19] proposed a number of static and dynamic diagram types for the seven identified views, see Table 4. He then discussed a small selection of methodologies that supported each diagram type, the methodologies being selected from over 20 contemporary AOSE methodologies (Table 7)—excluding those dealing with mobility, for example, Hachicha et al. [86], security (Low et al. [87] discuss security diagrams, offering them as extensions to existing diagrams—as shown here in Figure 4), for example, Mouratidis [85], and Bresciani et al. [66], or with noncooperative and adaptive agents. (We, however, do include aspects of ADELFE relevant to cooperative agents), for example, Georgé et al. [88] and Steegmans et al. [89], which introduce additional specifically-focused concepts, symbols, and diagram
types. Furthermore, Tran and Low [51] note that all are
deficient in at least one of the three areas of agent internal
design, agent interaction design, and MAS organization
modelling. The numbers for each diagram type proposed in
each of the methodologies of Table 7 are given in Table 8,
although it should be noted that some diagram types could
be classified under different headings.

In determining to which view (of Table 3) any specific
methodological diagram type should be allotted, terminology
definitions were sometimes found to be absent, ambiguous,
or apparently contradictory. There are several sets of such
terms including (i) organization and domain, (ii) interaction
diagram and protocol diagram, (iii) goal and task, and (iv)
“capability,” “service,” “responsibility,” and “functionality”.

Since some authors are using their own definitions, for
example, in categorizing views/perspectives, the scoping we
have established in Table 3 is sometimes not matched by par-
ticular methodological approaches. In particular, our antic-
ipation that the Architecture view should be independent of
technology chosen for the solution, as described, for instance,
in Giorgini et al. [67], is not met (see further discussion in
Section 5.2). In other methodologies, the different use of
terms such as “model,” “diagram,” “view,” and “viewpoint”
is often unclear (e.g., [20, 29, 31, 39, 49, 90]). As another
example, PASSI confounds work product terms with process
terms by using model/diagram names to describe tasks.

Another challenge in developing a standard diagramming
suite, useful for all AOSE methodologies, is that, while
some published methodologies recommend a set of diagrams
that occurs in every publication (e.g., [17, 44, 45]), other
methodologies continue to evolve so that examination of any
one methodology-specific paper often results in difficulty in
our determining of what diagrams are recommended for that
particular methodology at the present time, although some
authors do make it clear what changes have been made (e.g.,
[62]). In other words, some methodologies contain a stable
set of work products, whilst in others the recommended
diagramming suite has not yet stabilized.

While Henderson-Sellers [19] attempted to be compre-
prehensive, here we will emphasize those diagram types and dia-
gram usages recommended therein, extending the discussion
and incorporating new ideas on AOML notations [5]. When
standard UML (OO) diagrams are recommended, we will
not include a pictorial representation of what (we assume)
will be a diagram well known to readers, being part of the
International Standard 19501 [93].

We do not undertake a side-by-side methodology com-
parison, as is done, for example, in Tran and Low [94] or,
more recently, in Dam and Winikoff [95]. Rather, we try to
exemplify some of the differences in representational style for
diagrams pertinent to each of the several views identified in
Henderson-Sellers [19] and summarized below.

In the following two sections, we analyze diagram types
currently used in a number of AOSE methodologies using the
framework of Table 4. Section 5 discusses the various
static diagram types and Section 6 the dynamic counterparts.
For both sections, we adopt the seven views deduced in
Henderson-Sellers [19] and summarized below.

In Table 7: Prime references for the AOSE methodologies quoted here.

<table>
<thead>
<tr>
<th>Methodology name</th>
<th>Main references</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADELFE</td>
<td>Picard and Gleizes [22]</td>
</tr>
<tr>
<td>Agent factory</td>
<td>Collier et al. [23, 24]</td>
</tr>
<tr>
<td>CAMLE</td>
<td>Shan and Zhu [25]</td>
</tr>
<tr>
<td>Cassiopeia</td>
<td>Collinot et al. [26]</td>
</tr>
<tr>
<td>Elammari and Lalonde</td>
<td>Elammari and Lalonde [28]</td>
</tr>
<tr>
<td>Gaia</td>
<td>Wooldridge et al. [29]</td>
</tr>
<tr>
<td>ROADMAP extensions to Gaia</td>
<td>Juan et al. [32]</td>
</tr>
<tr>
<td>INGENIAS</td>
<td>Pavón and Gómez-Sanz [34]</td>
</tr>
<tr>
<td>ISLANDER</td>
<td>Sierra et al. [35, 36]</td>
</tr>
<tr>
<td>MAS-CommonKADS</td>
<td>Iglesias and Garijo [37]</td>
</tr>
<tr>
<td>MaSE</td>
<td>Wood and DeLoach [40]</td>
</tr>
<tr>
<td>O-MaSE</td>
<td>Garcia-Ojeda et al. [45]</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Caire et al. [47, 48]</td>
</tr>
<tr>
<td>MOBMAS</td>
<td>Tran et al. [50]</td>
</tr>
<tr>
<td>OperA</td>
<td>Dignum [52]</td>
</tr>
<tr>
<td>PASSI</td>
<td>Burrafato and Cossentino [54]</td>
</tr>
<tr>
<td>Prometheus</td>
<td>Padgham and Winikoff [59, 60]</td>
</tr>
<tr>
<td>RAP/AOR</td>
<td>Taveter and Wagner [17]</td>
</tr>
<tr>
<td>SODA</td>
<td>Omicini [63]</td>
</tr>
<tr>
<td>SONIA</td>
<td>Alonso et al. [64]</td>
</tr>
<tr>
<td>Tropos</td>
<td>Bresciani et al. [65, 66]</td>
</tr>
<tr>
<td></td>
<td>Giorgini et al. [67]</td>
</tr>
</tbody>
</table>

In determining to which view (of Table 3) any specific
methodological diagram type should be allotted, terminology
definitions were sometimes found to be absent, ambiguous,
or apparently contradictory. There are several sets of such
in Section 5.3, serves a second purpose: that of including not only the agent organization but also its interface with the environment, whilst retaining the (perhaps confusing) name of “organizational diagram.” This is especially seen in methodological approaches such as MAS-CommonKADS and MESSAGE. For the organizational model of the former, it is clear that the organization model is intended to serve also beyond the agent organization and to interface with the environment, since the recommended notation for the organizational model (Figure 6) includes sensors and actuators (a.k.a. effectors) in the agent symbol (actually an agent type—see earlier discussion).

In other words, some of the diagram types discussed in Section 5.1 could well be equally allocated to Section 5.3 (and vice versa). (For a more detailed and more philosophical discussion of environment abstractions, see Viroli et al. [99]). Environment was also recently a major topic of conversation within the OMG as part of their emerging interests in agents [100].

For the Architecture View, we note that the term “architecture” can have many interpretations in the context of an MAS. Here, we use it to describe large-scale features that are independent of the technology used to undertake the implementation of the MAS. In different AOSE methodologies, the level of detail can vary—some diagrams include agents and their roles whilst others do not.

Both the Environment View and the Architecture View diagrams are restricted to static diagram types.

In the Agent Societies View, diagrams depict agent societies or organizations. (As noted earlier, the term organization can be used both as a synonym for society and to represent the environment); for example, Ferber and Gutknecht [102] provide more detail than that of an architecture diagram. They typically focus on agent interaction...
Figure 5: Agents in an MAS interact with their environment using sensors and effectors.

Figure 6: Organizational model notation for MAS-CommonKADS (based on [37], reprinted by permission of the publisher © IGI Global).

rather than system structure (e.g., [63, 103]). Indeed, the architectural diagrams identified in Section 5.2 for various AOSE methodologies can also be extended to depict agent society details.

Furthermore, “organizational patterns” (i.e., patterns applied to agent societies) are discussed in Zambonelli et al. [30] and Gonzalez-Palacios and Luck [104]. Typical examples include pipeline, single hierarchy, and multiple hierarchies.

Here, we seek to depict how agents interact in terms of such an interacting society of agents and/or roles, again dividing the discussion into static and dynamic aspects.

The Agent Workflow View relates solely to dynamic diagram types since a workflow reflects agent behaviour. This can involves concepts such as process, actions, and interagent messaging.

For the Agent Knowledge View we need to represent the internal structure and behaviour of individual agents. Concepts such as goals, beliefs, commitments, plans, capabilities, perceptions, protocols, events, sensors, actuators, and services are all considered by one or more authors. In Section 5.4 we focus particularly on goals, ontologies, and plans.

The Agent Services View can involve a number of different diagramming techniques (see Table 4) including goals, tasks, capabilities, and a domain ontology. Services can be described as encapsulated blocks of offered functionality [30, 32, 105]. In AOSE, a service may be described in terms of capabilities, where a capability is defined as “the ability of an actor of defining, choosing and executing a plan for the fulfillment of a goal, given certain world conditions and in presence of a specific event” [65], a definition similar to that used in Prometheus.

For the Deployment View, the allocation of software components to hardware nodes has traditionally been the focus; for AOSE a greater emphasis is placed on agent conversations.

Finally, the UI View is ill represented in current AOSE methodologies. Our discussion therefore makes suggestions from outside the agent-oriented methodology community.

In the following two sections, citations to specific methodologies will be by methodology name rather than author name(s)—these are found in Table 7—unless a specific paper needs a direct citation. We introduce methodology-specific examples of diagram types not discussed in Henderson-Sellers [19] and assess their match to the previous recommendations. We also describe a selection of these diagrams with the new FAML notation [5], merely as an illustration of the visualization resulting from the combination of a specific diagram type and this notation. We introduce an oversimplified running example in the Travel Agent domain. None of these diagrams are intended to be a complete depiction but rather should be regarded as merely illustrations of the diagramming style to which they refer.

5. Static Diagram Types

5.1. Environment View. For an MAS, the environment is relevant to two separate phases of the development lifecycle.
Initially, requirements will relate to real-life problems, and the MAS will itself interact with this environment. This interaction will be evident in both the analysis and design phases. Secondly, environment issues are relevant in the deployment phase, when allocation of software code to a specific run-time platform node is necessitated. This second interface occasion is described in Section 5.6.

The recommended diagram type [19] for the environment description diagram, which models the external environment, is a UML-style class diagram with entities representing domain entities. For the environmental connectivity diagram, which shows the interfacial linkages between the environment and the top level agents in the MAS, particularly in terms of how agents are likely to access external resources such as databases, actors, and other MASs, a UML-style class diagram can also be useful. A third diagram type (more optional) is an External organization structure chart: a UML-style class diagram with entities = organizational unit, decomposition using the membership relation and acquaintance relationships between collaborating organizational units (see, e.g., Figure 7, which shows the use of this style of diagram in MOBMAS).

Environment description diagrams are also used in SODA and PASSI. INGENIAS offers an Environment Viewpoint diagram (Figure 8) depicting the external entities with which the MAS-to-be-constructed will interact.

A second style of diagram is often used to describe the functionality aspects relevant to the interaction between external stakeholders and the software system. This often relates to an early stage in the lifecycle, when requirements need to be identified and documented. Here, it is fairly common practice to use some sort of use case diagram, identical or very similar to that proposed in UML [10]. Henderson-Sellers [19] recommends that, to appropriately support the agent aspects more accurately, a goal-based use diagram that extends the “User-Environment-Responsibility (UER) case” diagram of Iglesias and Garijo [106] is useful for showing agent actors as well as human actors. An example of this is shown in Figure 9. To accompany this, a set of completed use case templates is necessary, such as that provided in Prometheus or by Taveter and Wagner [17], as originally proposed by Cockburn [107] (see example in Table 9). Here, the internal and external actors correspond directly to internal and external agents in AOR modelling.

Table 9: Example of a goal-based use case, here for the business process type “Process the request for a quote” (after [17], reprinted by permission of the publisher © IGI Global).

<table>
<thead>
<tr>
<th>Use case 1</th>
<th>Process the request for a quote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal of the primary actor</td>
<td>To receive from the seller the quote</td>
</tr>
<tr>
<td>Goal of the focus actor</td>
<td>To provide the buyer with the quote</td>
</tr>
<tr>
<td>Scope and level</td>
<td>Seller, primary task</td>
</tr>
<tr>
<td>Success end condition</td>
<td>The buyer has received from the seller the quote</td>
</tr>
<tr>
<td>Primary actor</td>
<td>Buyer</td>
</tr>
<tr>
<td>Secondary actors</td>
<td></td>
</tr>
<tr>
<td>Triggering event</td>
<td>A request for a quote by the buyer</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Step</td>
<td>Action</td>
</tr>
<tr>
<td>1</td>
<td>Check and register the availability of the product items included in the request for a quote</td>
</tr>
<tr>
<td>2</td>
<td>Send the quote to the buyer.</td>
</tr>
</tbody>
</table>

As is the case with the use of use cases in object-oriented software development, the use case diagram only offers a high level viewpoint on requirements. Of more value [107] is the textual description of each use case. In the Prometheus approach, Padgham and Winikoff [59] note that, since agents have abilities beyond those of objects, it is necessary to provide a textual template significantly beyond those found in OO requirements engineering. Specifically, their textual template (called a “functionality descriptor”) describes the system functionality in terms of name, description, percepts, actions, data used/produced, and a brief discussion of interactions with other functionality. While these functionality descriptors are said to be intermediate work products, a final work product that is cross-checked (Figure 10) with them is the use case scenarios (or “scenarios” for short). These are again textual—a typical scenario descriptor in Prometheus is given in Table 10. Each step described in the scenario is a small piece of functionality.

Other methodologies use UML use cases “as-is,” for example, MaSE, ROADMAP, ADELFE, MAS-CommonKADS and PASSI where it is called a “domain descriptor diagram.”

5.2. Architecture View. Henderson-Sellers [19] recommends a UML-style package diagram as the Organization-based architecture diagram (Figure 11) similar to that used in MESSAGE [49] (Figure 12), although this diagram often has a different name, using different basic shapes, for example, organization structure model in Gaia [29] (Figure 13), or as a jurisdictional diagram (as in Figure 14).

In INGENIAS, the generic elements of Figure 15 are used to depict an exemplar organizational viewpoint model in Figure 16 (notational key is given in Figure 17).
5.3. Agent Societies View. A large number of AOSE methodologies have a strong focus on agent societies, especially SODA, ISLANDER, and OperA. Social structures were added to the earlier version of Gaia by Zambonelli et al. [30].

The style of an agent society diagram recommended in Henderson-Sellers [19] is that of a UML-style class diagram showing all agents and all interrelationships. Other information that may be chosen for display includes percepts, actions, capabilities, plans, data, and messages represented by entities rather than relationships. An example is seen in Figure 18, which uses Prometheus notation and depicts individual messages in the style of Padgham and Winikoff [60], for example, their Figure 5 (p. 123). An alternative depiction is given in Figure 19, which gathers messages into interaction protocols, following the style of Padgham and Winikoff [60], for example, their figure in page 93; part of which is also represented with FAML’s notation in Figure 20.

Other diagram styles can be seen in, for instance, MASCommonKADS (Figure 21), which shows the mental state and internal attributes of an agent (e.g., goals, beliefs, and
plans) (upper box) together with the external attributes of the agent (e.g., services, sensors, and effectors) (lower box) (Figure 6); MASE (Figure 22), in which the connections between classes denote conversations that are held between agent classes, and the second label in each agent class represents the role the agent plays in a conversation; MOB-MAS (Figure 23), which shows acquaintances between agent classes and connections between these and any wrapped resources, a diagram that may also be enhanced to show protocols and associated ontologies; AOR (Figure 24), which

Figure 9: Recommended diagram for an agent goal-based use case diagram, (reprinted from [19], copyright 2010, with permission from IOS Press).

Figure 10: Phases and work products defined in Prometheus (after [60], reprinted by permission of the publisher © IGI Global).

Figure 11: Recommended diagram style for an Architecture Diagram.
Figure 12: Organization-based architecture diagram of MESSAG (after [49], reprinted by permission of the publisher © IGI Global).

Table 10: Example of a Prometheus scenario descriptor (after [60], reprinted by permission of the publisher © IGI Global).

<table>
<thead>
<tr>
<th>No.</th>
<th>Type</th>
<th>Name</th>
<th>Functionality</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Percept</td>
<td>Request meeting</td>
<td>User interaction</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Goal</td>
<td>Propose time meeting user preferences</td>
<td>Meeting scheduler</td>
<td>MeetDB(R),Prefs(R)</td>
</tr>
<tr>
<td>3</td>
<td>Goal</td>
<td>Negotiate with other users</td>
<td>Negotiator</td>
<td>MeetDB(R),Prefs(R)</td>
</tr>
<tr>
<td>4</td>
<td>Goal</td>
<td>Update user's diary</td>
<td>Meeting manager</td>
<td>MeetDB(W)</td>
</tr>
<tr>
<td>5</td>
<td>Goal</td>
<td>Inform others of meeting</td>
<td>Contact notify</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Other</td>
<td>Wait for day of meeting</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Goal</td>
<td>Remind user of meeting</td>
<td>User notify</td>
<td>MeetDB(R),Prefs(R)</td>
</tr>
<tr>
<td>8</td>
<td>Goal</td>
<td>Remind others of meeting</td>
<td>Contact notify</td>
<td>ContactInfo(R)</td>
</tr>
</tbody>
</table>

Variations:
(i) Steps 2–3 may be repeated in order to obtain agreement.
(ii) If agreement on a meeting time is not reached then steps 4–8 are replaced with notifying the user that the meeting could not be scheduled.
(iii) The meeting can be rescheduled or cancelled during step 6 (waiting).

Key:
(i) MeetDB(R): meetings database read.
(ii) Prefs(R): user preferences read.
(iii) MeetDB(W): meetings database written.
(iv) ContactInfo(R): contact information read.

Another approach to agent societies is to utilize a version of the UML collaboration diagram, although the omission of any sequencing of communications makes this use somewhat dubious. Typically, they depict static aspects of the agent society rather than being dynamic interaction diagrams (as depicted agent types, their internal agents, and the relationships between them; PASSI (Figure 25); and ADELFE (Figure 26), which depicts the connectivity between cooperative agents, for which ADELFE was specifically designed (Figure 26).
any true variant on a UML collaboration would be classified). Hence, they are summarized in this subsection.

The event flow diagram of MAS-CommonKADS [39], for example, represents events (the exchange of messages) (Figure 27) but does not depict any message sequencing. A somewhat similar diagram is to be found in OperA (Figure 28).

Visually different are the interaction-frame diagrams of RAP/AOR. This is used at both the class level (Figure 29) and the agent-instance level (Figure 30). In these diagrams, the
solid arrows indicate a noncommunicative action event type, and the chain dashed arrows are message types.

Whilst not a methodology, MAS-ML [71] suggests, in this context, the use of two diagrams that have a UML style class diagram to them: (i) the organization diagram and (ii) an extended UML Class Diagram that shows the “structural aspects of classes, agents, organizations, environments, and the relationships between these entities,” by introducing additional concepts (i.e., additional to those of UML Class Diagrams), including Environment Class,
Get location preferences
Get cost constraints
Get preferred mode of travel
Choose travel agent

UI of software system

Holiday maker

Bank transaction

Book holiday

Provide quote

Request cost

Request location information

Customer DB

Holiday info DB

Email total cost

Email booking receipt

Request availability

Holiday booking

Wholesale holiday supplier

Travel agent

Wholesale holiday purchase

Bank transaction

Bank transaction

Bank transaction

Holiday booking protocol

Holiday booking protocol

Customer database

Holiday information database

Travel agent

Wholesale holiday supplier

Figure 18: Prometheus style “system overview diagram” depicting messages.

Figure 19: Prometheus style “system overview diagram” depicting protocols.

Figure 20: A portion of Figure 19 “translated” into FAML notation.
5.3.1. Roles. Although supported to some degree in object-oriented modelling languages, the much greater importance of roles in AOSE modelling [109–113] requires separate consideration, despite the common adoption of a UML-style class diagram to depict roles (Figure 32), for example, as used in Agent Factory, MOBMAS, PASSI, and O-MaSE. It should be noted that, in Figure 32, each role class is characterized by its associated so-called protocol identifiers, while each agent class is characterized by a list of protocol identifiers (any not specified in the associated role) and activities. Consequently, it is argued that a two-compartment symbol is needed for roles, and a three-compartment symbol for agents. In ROADMAP also, roles are explicitly linked to agents, as shown in Figure 33 [114]. Such role-focussed diagrams should be supplemented by a Role definition template (see, e.g., Table 11). The original ROADMAP template [115] was later simplified by Sterling et al. [33] as shown in Table 12.

A second role-focussed diagram is the role dependency diagram: a simple role decomposition diagram, using just names and unadorned lines. Figure 34 is one such example. This may be supplemented by a textual description of all the roles and their dependencies (Table 13).

Whilst not a methodology, MAS-ML [71] uses a UML-like Role Diagram using the notation given in Figure 31 to show “structural aspects of agents roles and object roles defined in the organisations” and their interrelationships. Concepts additional to those of UML Class Diagrams include Object Role Class to represent resources utilized by an Agent Role Class.
**Figure 24:** AOR agent diagram for the domain of B2B e-commerce (after [17], reprinted by permission of the publisher © IGI Global).

**Figure 25:** PASSI’s Multiagent Structure Definition Diagram.

**Table 11:** Role definition as used in Opera (after [52]).

<table>
<thead>
<tr>
<th>Role: PC member</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objectives:</strong></td>
</tr>
<tr>
<td>Paper_reviewed(Paper, Report)</td>
</tr>
<tr>
<td><em>Subobjectives:</em></td>
</tr>
<tr>
<td>{read(P), report_written(P, Rep), review_received(org, P, Rep) }</td>
</tr>
<tr>
<td><strong>Rights:</strong></td>
</tr>
<tr>
<td>access-confman-programme(me)</td>
</tr>
<tr>
<td><strong>Norms:</strong></td>
</tr>
<tr>
<td>OBLIGED understand(English)</td>
</tr>
<tr>
<td>IF DONE assigned(P, me, Deadline) THEN OBLIGED paper_reviewed(P, Rep) BEFORE Deadline</td>
</tr>
<tr>
<td>IF DONE paper_assigned(P, me, _) AND is_a_direct_colleague(author(P)) THEN OBLIGED review_refused(P) BEFORE TOMORROW</td>
</tr>
<tr>
<td><strong>Type:</strong></td>
</tr>
<tr>
<td>External</td>
</tr>
</tbody>
</table>

Figures 35 and 36 show how the role dependency diagram of Figure 34 and the Agent-role dependency diagram of Figure 33 can be depicted using the FAML notation of Figure 4.

5.4. Agent Knowledge View. The static knowledge of individual agents is encapsulated in symbols for each agent type, typically by extending a basic icon, such as the UML rectangle or the MOSES tablet (as used, for instance, in Figure 21). Suggestions here include goals, beliefs, commitments, and plans added to the basic class symbol (Figure 37); however, several authors (e.g., [76]) argue that it is not yet clear whether the attributes and operations are valid features of an agent type. Since these are derived (via the metamodel) from the UML Classifier, their rejection would negate the generalization relationship between Agent and Classifier (as shown in Figure 36). (This is another illustration of the confounding in the literature between agent and agent type. Often what is referred to as an agent is an agent type, that is, the word is used to describe an entity that conforms to some subtype of Classifier in the metamodel. Although in our following analysis we will continue to use “agent” when quoting from the AOSE literature, it should be remembered that usually this should be replaced by “agent type”).

Thus this suggests the need for an agent modelling language *not* based on UML (see revisions, proposed here, in Figure 38). Other proposals are to explicitly depict
Figure 26: Typical ADELFE class diagram showing two cooperative agents and their interrelationships with other classes.

Figure 27: Event flow diagram using MAS-CommonKADS notation.

Table 12: Simplified version of the role template as used in more recent versions of ROADMAP—here for an Intruder Handler agent (reprinted from [33], copyright 2006, with permission from IOS Press).

<table>
<thead>
<tr>
<th>Role name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intruder handler</td>
<td>Identifies and responds to the intruder detected</td>
</tr>
<tr>
<td>Responsibilities</td>
<td>Detect the presence of a person in the environment</td>
</tr>
<tr>
<td></td>
<td>Check the house schedule for strangers scheduled to be there</td>
</tr>
<tr>
<td></td>
<td>Take an image of the person</td>
</tr>
<tr>
<td></td>
<td>Compare the image against the database of known people</td>
</tr>
<tr>
<td></td>
<td>Contact the police and send the image to them</td>
</tr>
<tr>
<td></td>
<td>Check the house schedule for planned visitors</td>
</tr>
<tr>
<td></td>
<td>Send a message to stay away to each visitor expected that day</td>
</tr>
<tr>
<td></td>
<td>Inform the owner that the police are on their way and the visitors</td>
</tr>
<tr>
<td></td>
<td>have been warned not to enter the house</td>
</tr>
</tbody>
</table>

Constraints

- The owner and each person pointed out by him/her needs to provide in advance personal information (face) to be recognised
- A subject to be detected needs to be seen within the camera's image area
- The user must maintain the schedule
- Visitors must be within the coverage area of mobile communication with their mobile access terminals switched on

Table 13: Social structure definition in OperA (after [52]).

<table>
<thead>
<tr>
<th>Social structure definition</th>
</tr>
</thead>
</table>

**Roles:**

A list of role definitions

**Role dependencies:**

A list of triples of two role names and the name of the relationship between them

**Groups:**

A list of sets of roles

Capabilities, perceptions, protocols, and organizations (Figure 39); belief conceptualization and events (Figure 40); or sensors, actuators, and services (Figure 6); these often being supplemented by agent descriptors (see, e.g., Boxes 1 and 2). Textual templates for agent roles are recommended in Gaia [29]. Such a schema (one per role) gives information about the protocols, permissions, and responsibilities (liveness and safety) as well as an overall description for each agent role. It is also used by Suganthy and Chithralekha [118] and in SODA.

Knowledge is often expressed in terms of the “mental state” (Figure 41) of an agent, as described, for instance, in the Agent viewpoint of INGENIAS [20]—see example in Figure 42. This idea of “mental state” is also found in AOR (Figure 43) and in Silva et al. [119] who link the set of beliefs, goals, plans, and actions to the mental state of the agent.

Agent internals are represented in Prometheus in terms of an Agent Overview diagram (one for each agent). For the Travel Agent agent in Figure 19, Figure 44 shows its capabilities, percepts, and messages utilized. It is similar in style to the System Overview diagram of Figure 19 but at a finer granularity (Figure 45). Then each capability in the Agent Overview diagram can be expanded in a Capability Overview diagram (Figures 46 and 47).

In SONIA, the knowledge model is represented differently, by blocks of knowledge that group concepts and
5.4.1. Goals. Another aspect of agent knowledge is that of goals: agent goals as well as system goals. A goal is said to represent a state that is to be achieved, for example, Braubach et al. [120]—although other kinds of goals are possible, and goals may conflict with each other, for example, Van Riemsdijk et al. [121]. Goals are achieved by means of actions (a.k.a. tasks) (Figures 48 and 49), the combination of
Figure 29: AOR interaction frame between agents of the types Buyer and Seller (after [17], reprinted by permission of the publisher © IGI Global).

Figure 30: AOR interaction sequence between agent instances BuyerA and SellerB (after [17], reprinted by permission of the publisher © IGI Global).
actions and goals forming the plan body. Tasks are discussed later in Section 6.4.

Goal-focussed diagrams are found in Prometheus (Figure 50), in MaSE, and in MESSAGE’s “Level 0 Goal/Task Implication Diagram” (Figure 51). Prometheus also suggests a textual goal descriptor—with three lines only: name, description, and subgoals. (We note that in Figure 50, following Prometheus’ guidelines, the names of the goals are almost task-like (i.e., verbs). Here, from Figure 48, we argue for state-like names for goals, as shown in Figure 52).

Relationships between goals are captured in MOBMAS’s agent-goal diagram (Figure 53) and in the two goal-focussed diagrams of Tropos (Figures 54 and 55). The actor diagram (Figure 54) links actors to goals, whereas the goal
Figure 34: Role dependency diagram as used in OperA for the “Conference Society” (after [52]).

diagram (Figure 55) expands the internal details of the goal itself—here using the $i^*$ notation of Yu [78] which permits discrimination between hard goals and soft goals. (It can be determined whether or not a hard goal has been satisfied; in contrast soft goals do not have well-defined achievement criteria and can only be “satisficed” (e.g., [65, page 207]). Hard goals are associated with capturing functional requirements and soft goals with nonfunctional requirements, e.g., Braubach et al. [120].) The goals, labelled Gx in Figure 53, are also captured in the third partition of the Agent class diagram (Figure 40).

A diagram type that appears to have some ambiguity in terms of its scoping (system versus agent) is found in MaSE, called the “goal hierarchy diagram” (Figure 56) (called Goal Model in OMaSE), and in MESSAGE, where it is called an agent goal decomposition diagram linking goals, tasks, actions, and facts. In both methodologies, it is a relatively simple tree structure where goals are represented as boxes and goal-subgoal relationships as directed arrows from parent to children (MASE) or children to parent (MESSAGE). (Clearly, such directional contradictions form an ideal target for standardization.) It is interesting to observe that this would appear to be topologically isomorphic with Graham’s [122] task decomposition diagram (using hierarchical task analysis). Indeed, based on our earlier discussion, Figure 56 is, more realistically, either a task (not a goal) diagram or else is a goal diagram with poorly named goals.

Goal hierarchy diagrams are also used in Hermes [123] but associated with agent interactions (Figure 57), and a goal-oriented interaction modelling technique is also introduced into Prometheus by Cheong and Winikoff [124] to replace the previous interaction protocol modelling techniques used in Prometheus.

A more elaborated version of this approach, based on the well-established AND/OR approach to goal modelling, is shown in Figure 58, as presented in OMaSE. The numbers indicate a precedence ordering of the goals, again suggesting tasks rather than goals, although goals are, of course, closely linked to tasks (see Section 6.4), as shown in the metamodel of Figure 48.

Notwithstanding, this pair of concepts (“task” and "goal") are frequently confused. In an etymological analysis of these terms, Henderson-Sellers et al. [101] recommend goals as future-desired states that, when committed to, require the enactment of a task (sometimes called “action”) in order to achieve such a desired state (Figure 48). Thus the enactment of a task requires a duration. At the point of time at which this ends, the goal has been achieved (Figure 59). This means that goal names should be state names: that is, nouns, whereas task names should be more verb-like. Splitting up the achievement of a final goal into a set of intermediate or subgoals, as shown here, permits a differentiation (goal/subgoal) that could be seen as commensurate with the action/task differentiation of Figure 60; that is, a goal is achieved by an action, which can be broken down into more granular sections each of which depicts a subgoal being achieved by a task. However, in some methodologies these two terms (goal and task) are equated; that is, used as synonyms. This can often lead to names that are cognitive misdirectional, for example, in such methodologies, the names of goals are typically imperative
Figure 35: Role dependency diagram depicted using the FAML notation.

Figure 36: Agent-role coupling diagram depicted using the FAML notation.
verb-like which, at first glance, suggests tasks rather than goals. This, therefore, needs to be borne in mind when reading or writing such diagrams.

5.4.2. Ontologies and Plans. Also in this group of recommendations (for static diagram types relevant to agent knowledge) are the ontology diagram and the plan diagram.
### Agent class

<table>
<thead>
<tr>
<th>Name/rolename</th>
<th>Belief conceptualizations</th>
<th>Agent goals</th>
<th>Events</th>
</tr>
</thead>
</table>

**Figure 40:** Agent class definition diagram of MOBMAS.

**Figure 41:** Example of an agent's mental state (which links mental state to facts, beliefs, and events and involves goals, tasks, and roles) as depicted by INGENIAS (after [20], reprinted by permission of the publisher © IGI Global).

**Figure 42:** Typical elements in the agent viewpoint and the agent's mental state as depicted by INGENIAS (after [20], reprinted by permission of the publisher © IGI Global).

**Figure 43:** Core mental state structure modelling elements of external AOR diagrams (after [17], reprinted by permission of the publisher © IGI Global).
Ontologies are explicit in only a handful of methodological approaches: PASSI and MAS-CommonKADS, MOBMAS and AML [105], and, to a lesser extent, in OperA and ISLANDER. Ontologies, particularly domain ontologies as need here, represent knowledge that is effectively static. It is thus reasonable to depict that knowledge as a fairly standard UML class diagram (Figure 61).

Plans depict the internal details of how a task is to be performed and a goal attained.Plans are typically internal to a single agent and are linked to the tasks (or actions) needed to attain goals (Figure 48) or to the capabilities (Figures 44 and 46). Since the execution of plans may or may not be successful, alternative paths must be included (Figure 62). These alternatives may utilize AND/OR gates, which can be used either in context of an activity diagram or a state transition diagram—depending upon whether the developer wishes to have as his/her prime focus the process or the product aspect. In Tropos, the internal structure of a plan can be summarized as a single node on a Capability diagram (e.g., Figure 46). Plan diagrams may be based on the UML activity diagram as in Tropos or UML STD diagram as in O-MaSE. Mylopoulos et al. [79] show how the Tropos plan diagram can also be depicted using UML notations. Plan diagrams are also used in MOBMAS.
Plan name: Identify location and dates
Description: Ascertain possible holiday places and date
Trigger: Request from client
Context: Holiday not already booked
Data used and produced: Holiday brochures/database
Goal: Recommend time and place(s)
Failure: All likely holidays booked
Failure recovery: Change dates and/or place
Procedure: (1) Search data for location commensurate with client’s desire
(2) Check dates holiday is possible
(3) Create list of possible place/date combinations

Box 3: Prometheus-style plan descriptor for the plan to identify holiday location and dates of Figure 46.

Plan diagrams, whether of the activity diagram style or the STD style, can be augmented by text in the form of a plan descriptor (Box 3), as used, for example, in MOBMAS, which defines the plan in terms of initial state, goals, strategies, actions, and events, and Prometheus, which defines a plan in terms of triggering events, messages, actions, and plan steps (a completed example of which is shown in Box 3).

Thangarajah et al. [125] note that there may be several acceptable plans for achieving a single goal such that there is an overlap. This led these authors to formulate mathematical
Figure 48: Metamodel fragment relevant to goals, tasks, and plans (after [101]).

Figure 49: Generic model of plans, tasks, and goals conformant to a fragment of the metamodel of Figure 48.

Figure 50: Example Prometheus Goal Overview Diagram.
expressions for this overlap and also the coverage. Overlap is readily represented using a Venn diagram; a typical goal-plan hierarchy is shown in Figure 63.

5.5. Agent Services View. UML-style class diagram is supplemented by UML-style activity diagrams to show details for each capability to expand a portion of the Capability
Figure 54: (a) Example actor diagram showing goals attached to actors. (b) Example of actor diagram showing an explicit depender (Holidaymaker), dependee (Travel Agent), and dependum (Select good travel agent).

Figure 55: Example of a Tropos goal diagram.

Figure 56: Goal hierarchy diagram (MaSE).
Overview diagram of Figure 46 into a more detailed Capability diagram—one diagram for each subcapability. An example, compatible with the Check availability capability of Figure 64, is given in Figure 65 (Prometheus notation). A textual template to accompany the diagram might also be useful to present in textual format details of goals, processes protocols, messages, percepts, actions, capabilities, plans, and data utilized in different ways. Figure 66 shows the alternative use of an Activity Diagram in this context. Capability Diagrams being used in Tropos (Figure 66 for a specific agent) wherein each node may be expanded into a Plan Diagram (see Section 5.4.2).

Services can alternatively be represented directly in either graphical or tabular form, the latter following, for example, the Gaia Service Model, which lists Services and their Inputs, Outputs, Preconditions and Postconditions, the former as depicted in the Level I analysis phase of MESSAGE [48, page 186], wherein a service is realized by a partially ordered set of tasks (see later discussion of Figure 81). Direct representation of service protocols is found also in AML (see later discussion of Figures 76 and 77) and in the Service Model of ISLANDER [35].

5.6. Deployment View. Henderson-Sellers [19] recommends using a fairly standard UML (or AUML) deployment diagram. MaSE uses a similar diagram (Figure 67) but notes that it differs from the standard use of the UML Deployment Diagram since

(i) the three-dimensional boxes represent agents in MASE, whereas they represent (hardware) nodes in UML,
Figure 59: Milestones, subgoals, and goals: (a) a single action attains the goal or (b) several actions are needed, each achieving a subgoal (after [101]).

Figure 60: The action perspective of the metamodel of Azaiez et al. [108], (reprinted from [108], copyright 2007, with permission from IOS Press).

(ii) the connecting lines represent conversations between agents in MASE whereas in UML they represent physical connections between nodes,

(iii) MASE uses dashed-line box around agents to indicate that these agents are housed on the same physical platform.

Other methodologies adopting this UML style of deployment diagram include PASSI and RAP/AOR. Most other approaches neglect it. Whilst not employing such a diagram, Tropos does discuss implementation issues as related to its use of class diagrams (Figure 68). SODA only hints at implementation in terms of their environment model, preferring to
Figure 61: Ontology diagram as used in MOBMAS.

Figure 62: Plan diagram in which \(ac\) is the activation condition and \(\alpha\) is the activation action. Stop states are labeled as success states “✓” (success action “\(\sigma\)”), fail states “×” (fail action “\(\varphi\)”), unknown states “?” (unknown action “\(\nu\)”), or abort states “A” (abort condition “\(\omega\)”; abort action “\(\omega\)”) (after [116], reprinted by permission of the publisher © IGI Global).

Figure 63: An example goal-plan hierarchy diagram using the notation proposed by Shapiro et al. [117].
5.7. UI View. Henderson-Sellers [19] noted the lack, in published AOSE methodologies, of any diagrams relating to the user interface. He therefore recommended adding (at least) a
UI design diagram, which could likely be represented using a semantic net (Figure 69). Henderson-Sellers [19] offers this as a placeholder; that is, a generic “UI design” diagram type pending future empirical work and utilization of the visual design theories of, for example, Ware [127] and the insights of Graham [122] and Constantine and Lockwood [128] (see also http://www.foruse.com/). This latter book also recommends user role maps and structure role models (where “role” refers to human roles in the software development process). These authors also provide heuristics on menu design, the use of iconic interfaces, and other more innovative interface design approaches. The topic was also explored in a non-AOSE context by Gonzalez-Perez [129].

Other suggestions in the literature, although sparse, include a placeholder for a UI prototype in ADELFE’s (http://www.irit.fr/ADELFE/) Activities 8 and 9 (see also Jorquera et al. [130] who discuss UI prototyping as a work unit but do not offer a notation for the resultant work product).

6. Dynamic Diagram Types

6.1. Agent Societies View. Agent behaviour is usually depicted in terms of agent-agent interactions, including message passing. A typical agent-oriented interaction diagram also shows the order of these messages needed to effect a single service. Consequently, a standard AUML [75] or AML [73] interaction diagram can be used as the basic interaction diagram (a.k.a. conversation diagram) (Figure 70). Further addition of more formal protocol information to the basic conversation diagram, again using, say, AUML or AML as the notation, would result in a protocol diagram. Prometheus optionally enhances these interaction diagrams with percepts and actions; that is, messages to and from an invisible timeline/agent. In addition to percepts, input from the environment and other events, including those self-generated, for example, by a clock, can be shown [131]. Events typically generate an action within the agent. The result is really
Figure 68: Partial class diagram for a Tropos implementation in their Store Front case study (after [67], reprinted by permission of the publisher © IGI Global).

Figure 69: An OPEN/Metis user interface sketch (after [126]).
Agent Factory (Figure 73), where it is called a “protocol” and a “protocol model,” respectively, and in MESSAGE (Figure 74) and Tropos (Figure 75). MOBMAS uses a slightly different version of an AUML sequence diagram in which ACL messages are replaced by tuples.

A specialized form of the AML Interaction Protocol Diagram (Figure 72) is the Service Protocol Diagram (Figure 76), used only within the context of the service specification.

Whilst not a methodology, MAS-ML [71] uses an extended UML Sequence Diagram to show interactions and their sequencing. These authors use this approach to depict the modelling of plans and actions, of protocols, and of role commitment.

Collaboration-style diagrams are used in Agent Factory, CAMLE, and AML (Figure 77) but seldom elsewhere wherein the sequence-style diagram is the preferred option.

6.2. Workflow View. Workflows reflect agent behaviour so that a standard workflow diagram would seem appropriate. UML-style activity diagrams are used in Prometheus to illustrate processes within an agent, including allusions to
interactions with other agents. Figure 78 shows an example of a Prometheus Process Diagram. In this diagram, details of a process are shown together with interactions with other agents; these are depicted minimalistically by a single (envelope-shaped) icon. A Process Diagram can then be supplemented by a process descriptor listing activities, triggers, messages, and protocols. Whilst these diagrams show the higher level view, details can be presented textually.
Figure 74: Example of an MESSAGE interaction protocol diagram.

Figure 75: Example agent interaction protocol as used in Tropos.

[132]: for percepts, actions, and events. The percept descriptor lists the information gleaned by the agent in its interaction with the environment, whereas the action descriptor depicts the effect of the agent on the environment. An event descriptor defines an event in terms of its purpose, together with the data that the event carries. (For details of these templates, see [59, Chapter 7]).

UML-style activity diagrams are also used in Agent Factory, called there an "activity model." These illuminate all the "activity scenarios," wherein each swimlane represents the processing of a role involved in the scenario. In PASSI, a similar use is made of swimlanes to specify the methods of each agent or of each agent’s task.

Workflows are also represented explicitly in INGENIAS (Figure 79) using the notation shown in Figure 80. In this approach, a workflow is created from the tasks identified in the interaction specification diagram together with the goal/task view (see later discussion of Figure 84). Similarly,
MESSAGE depicts a workflow in terms of a partially ordered set of tasks that realize a service (Figure 81). AOR also uses workflow as the basis for its activity diagram (Figure 82).

MAS-ML [71] includes extensions proposed to the UML Activity Diagram in order to depict “a flow of execution through the sequencing of subordinate units called action.” In this way, the authors can model plans and actions; goals; and messages, roles, organizations, and environment (for full details, see [133]).

6.3. Agent Knowledge View. The dynamic aspects of agent behaviour are addressed in several methodologies, both in terms of interagent behaviour and single agent behaviour. This is exemplified in MaSE’s “communication class diagram” a.k.a. conversation diagram [43], based on the notation of a UML State Transition Diagram. It should be noted that this diagram type focusses on the states of an agent during a particular conversation. This means that for a conversation between two agents (initiator and responder) two state diagrams are required. Actions specified within a state represent processing required by the agent.

STD-style diagrams are recommended by PASSI, MES- SAGE (Figure 83), and MOBMAS. Also with a slightly different visualization is the state machine-focussed Interaction Structure of ISLANDER, which shows dialogues called scenes. These then define protocols.

6.4. Agent Services View: Task Diagrams. Tasks represent the dynamic counterbalance to goals. Indeed, they are closely linked, as shown in the metamodel of Figure 48 and in INGENIAS (Figure 84). Indeed, in the SONIA methodology, a task model is one of the first to be developed, although
goals are not considered until later, in a Goal Model.
Tasks are accomplished by the enactment of a Plan (see
Section 5.4.2).
Several AOSE methodologies address task descriptions
and task decomposition, for example, using a UML activity
diagram with two swimlanes (Figure 85) or incorporating
AND/OR task decomposition (Figure 86) to create a task
hierarchy. Both diagram styles can be supplemented by Task
templates. For the current state(s) of any task, a standard
UML-style Statechart diagram can be used.
A task model is also included in the diagram suite of MAS-CommonKADS. It consists of two elements for each task: a "task hierarchy diagram" and a "task textual template." Peyravi and Taghyareh [68] suggest the addition to MAS-CommonKADS of a standard activity diagram to represent the activity flow of a task together with a textual template for each individual task within the activity flow, together with a tabular rendering of task knowledge. Task models are also found in MaSE and in ISLANDER.

Interestingly, Fuentes-Fernández et al. [134] investigate the ideas of Activity Theory [135] as applied to AOSE. They propose mapping these activities to tasks and possibly to workflows or interactions. This is clearly a topic for further discussion beyond the standard methodological use of diagram types as outlined here.

6.5. UI View. In Section 5.7, we noted the possibility of introducing, as a static diagram type, a UI design diagram. There is also a need for a dynamic view on the UI. Gonzalez-Perez [126] suggests a service state diagram (Figure 87) to represent the various states of the UI and linking this directly to UI design diagram of Figure 69. Constantine
and Lockwood [128] also recommend the utilization of task modelling, essential use cases, and context navigation maps to describe the dynamic aspects of user interface design.

## 7. Discussion and Related Work

Recommendations for a standardized agent-oriented modelling language are still indeterminate with several approaches being investigated. These include use of many UML diagrams with little or no change (e.g., [20, 37, 49, 59, 136]) or bundled as a UML profile (e.g., [17, page 286]). Formal proposals to create an agent-oriented extension to UML include AUML [74, 137, 138] and AML [73]. They provide all the fine detail of the UML, also making some recommendations for diagram types in passing. Here, we have taken the results of the deliberations of Henderson-Sellers [19] regarding appropriate diagram types that could be recommended as part of a future standardized agent-oriented modelling language and have investigated a wider range of examples from the literature.

In addition, we have introduced the FAML notation in order to see whether (i) the recommended diagram types can be visualized using this notation and (ii) there are any deficiencies in the FAML notation.

Our discussion here has focussed solely on the suite of diagram types, whilst recognizing that there are two other major elements of any modelling language: notation of individual atomic elements (e.g., [2, 100]) and the defining metamodel (e.g., [1]). In the latter case, there have been attempts to combine existing metamodels synergistically for (a) work products (e.g., [4, 71, 139]) and (b) method elements (e.g., [3, 140–142]).

In the A&A (agents and artifacts) approach [143], the three basic categories identified of agent, society, and environment align well with three of the views presented here (Tables 3 and 4). The aim of these authors is to be able to manipulate agent societies and the MAS environment as first-class entities. Their utilization of a multidisciplinary approach including speech act analysis [144] and activity theory [135]...
Figure 84: Goal-task relationships in INGENIAS (after [20]). For notation see Figure 17, it is reprinted by permission of the publisher © IGI Global.

Figure 85: Example of PASSI task specification diagram.

Figure 86: MOBMAS’s use of a task decomposition diagram.
is a promising approach to gain a well-grounded conceptual foundation for agent-oriented modelling.

It should also be noted that all work products go through their own “lifecycle” in the sense that they are first created, and then modified towards maturity of a final state. This means that there will likely be several versions of each work product (e.g., [145, Appendix G]); that is, the notion of a “state” can be associated with each work product (e.g., [45, 146]).

As well as agent-oriented metamodelling treatises, some authors have sought to set their work in the context of model-driven engineering (MDE) or model-driven development (MDD), for example, Amor et al. [147]; Fischer et al. [148]; Taveter and Sterling [149] and the proposal by Benguria et al. [150] of Platform Independent Model (PIM) for Service-Oriented Architecture (SOA) named PIM4SOA. Others (e.g., Liu et al. [151]) have examined the possible utilization of agents for developing web services and in SOA (Service-Oriented Architecture). Internet development is also discussed by Zambonelli et al. [152].

There is also an emerging trend to adopt a method engineering mindset for agent-oriented software construction (e.g., [153]). In a recent paper on O-MaSE [46], the authors tabulate their recommended diagram types in the form of method fragments (Table 14).

In addition to these related works on notation and metamodelling, we were only able to find a small number of additional papers in the main topic area that are not already cited above. In particular, although differently named in part, the six diagram types proffered by Juan et al. [154] in their “skeleton methodology” are commensurate with those discussed here. They are given as Use-case model, Environmental Interface Model, Agent Model, Service Model, Acquaintance Model, and Interaction Model. In addition, they proffer (i) from Prometheus: System Overview Diagram, Agent Overview Diagram, Capability Diagram, and Event, Data, Plan, and Capability Descriptors; and (ii) from ROADMAP: Environment Model, Knowledge Model, and Role Model.

Our use of FAML notation has only been indicative rather than the provision of any conclusive results. We have anticipated following the comprehensive evaluation method of the notation for ISO/IEC 24744 International Standard [21] as undertaken by Sousa et al. [8]. However, it turns out that there are significant differences between the mode of utilization of symbols in creating a process model (for which ISO/IEC 24744 was designed) and the way symbols are used in an AOML. In the former, not only are the symbols evaluated but also, perhaps more critically, the superposition of various combinations in terms of their usability vis-a-vis their shape and colour turns out to be the more important aspect. On the contrary, for an AOML, there is little (or zero) need to superpose symbols rather than to have a collection of them related to each other in any specific diagram type. This means that a symbol set for a AOSE diagramming suite need have little concern for juxtapositioning and superpositioning issues but simply be evaluated in terms of its semiotic value in terms of the degree to which each symbol successfully represents each AOSE concept. That means that our illustration of only a few diagram types, chosen to illustrate elements of the different “families” of Figure 2, indicates that further one-to-one translation of symbols between, say, Prometheus and FAML or between INGENIAS and FAML should be as successful. In terms of the goals of this current paper, the profferings of FAML diagram types are adequate, whilst leaving to future work (Section 8) comprehensive user and usability studies.

Thus, creating a standard, for which this paper is intended to be a potential precursor, needs careful mappings between the various methodology-linked diagram types by consideration of their associated semantics (i.e., not just names and notations). Once these similarities and any overlaps have been identified, it is likely that the number of diagram types needed for AOSE can be significantly reduced from the sum
Table 14: Diagram types recommended in O-MaSE depicted as method fragments (after [46]) and reprinted by permission of the publisher © Inderscience.

<table>
<thead>
<tr>
<th>Activities</th>
<th>Tasks</th>
<th>Work products created or modified</th>
<th>Responsible method roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements gathering</td>
<td>Requirements specification</td>
<td>Requirements specification</td>
<td>Requirements engineer</td>
</tr>
<tr>
<td>Problem analysis</td>
<td>Model goals</td>
<td>Goal model</td>
<td>Goal modeller</td>
</tr>
<tr>
<td></td>
<td>Refine goals</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model domain</td>
<td>Domain model</td>
<td>Domain modeller</td>
</tr>
<tr>
<td>Solution analysis</td>
<td>Model organization interfaces</td>
<td>Organization model</td>
<td>Organization modeller</td>
</tr>
<tr>
<td></td>
<td>Model roles</td>
<td>Role model</td>
<td>Role modeller</td>
</tr>
<tr>
<td></td>
<td>Define roles</td>
<td>Role description document</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Define role goals</td>
<td>Role goal model</td>
<td></td>
</tr>
<tr>
<td>Architecture design</td>
<td>Model agent classes</td>
<td>Agent class model</td>
<td>Agent class modeller</td>
</tr>
<tr>
<td></td>
<td>Model protocols</td>
<td>Protocol model</td>
<td>Protocol modeller</td>
</tr>
<tr>
<td>Low level design</td>
<td>Model policies</td>
<td>Policy model</td>
<td>Policy modeller</td>
</tr>
<tr>
<td></td>
<td>Model plans</td>
<td>Agent plan model</td>
<td>Plan modeller</td>
</tr>
<tr>
<td></td>
<td>Model capabilities</td>
<td>Capabilities model</td>
<td>Capabilities modeller</td>
</tr>
<tr>
<td></td>
<td>Model actions</td>
<td>Action model</td>
<td>Action modeller</td>
</tr>
<tr>
<td>Code generation</td>
<td>Generate code</td>
<td>Source code</td>
<td>Programmer</td>
</tr>
</tbody>
</table>

of all the diagram types across all AOSE methodologies—the aim of this current project.

8. Conclusions and Future Work

Using the list of over two dozen proposed diagram types in the AOSE literature, we have here extended the analysis of Henderson-Sellers [19] commencing with his recommendations and assessing to what extent these recommendations are seen in this wide range of AOSE methodologies. We have also taken the opportunity, as indicated in the Future Work Section of Henderson-Sellers [19], to express several of these recommended diagram types using the new FAML notation [5], itself conformant to the metamodel of Beydoun et al. [4], and derived in part from the suggestions of Padgham et al. [2] and Deloach et al. [155], and taking into account the semiotic advice of Moody [7].

In summary, we have added further evaluations of the recommendations of Henderson-Sellers [19] by consideration of a wider range of diagram types in the AOSE methodologies listed in Table 7, the motivation being a small additional contribution to standards efforts current in organizations like FIPA, OMG, and ISO.

As noted in Henderson-Sellers et al. [5], further empirical research is required to evaluate the usability of these various diagram types using FAML’s notation. We plan to undertake an experiment in which creative design students are asked to supply appropriate symbols for the FAML concepts as well as evaluating our current proposals (Figure 2). We also intend to conduct a comprehensive evaluation using a large group (20 plus) of experts followed by a usability study in a real world case study.

Acknowledgments

The author wishes to thanks Cesar Gonzalez-Perez for helpful comments on an earlier draft of this paper. This is contribution 12/06 of the Centre for Object Technology Applications and Research at the University of Technology, Sydney.

References


