Verifying Multi-Agent Knowledge-Based Systems using COVERAGE*

Alun Preece and Neil Lamb†

University of Aberdeen, Computing Science Department
Aberdeen AB9 2UE, Scotland
Phone: +44 1224 272296; FAX: +44 1224 273422
Email: apreece@csd.abdn.ac.uk

Abstract

Anomaly detection, as performed by the COVER tool, has proven to be a useful method for verification of knowledge-based systems. The increasing development of distributed knowledge-based systems based upon the multi-agent architecture demands techniques for the verification of these systems. This paper describes the COVERAGE tool — an extension of COVER designed to perform anomaly detection on multi-agent systems. The paper includes an example of a multi-agent system verified using COVERAGE.

KBS and MAS

Knowledge-based systems (KBS) are a relatively mature aspect of artificial intelligence technology. These systems solve problems in complex application domains by using a large body of explicitly-represented domain knowledge to search for solutions. This approach enables them to solve problems in ill-structured domains which defeat more conventional algorithmic programming. Unfortunately, this approach makes KBS harder to validate and verify because there is not always a definite "correct" solution. However, in recent years considerable progress has been made in developing effective validation and verification (V&V) techniques for such systems (Gupta 1990). One verification tool in particular, COVER, has been demonstrated as an effective means of revealing flaws in complex KBS (Preece, Shinghal, & Batarekh 1992). COVER operates by checking the knowledge base of a KBS for logical *anomalies* which are symptoms of probable faults in the system.

Recently it has been realised that in order to solve certain kinds of complex problem it is necessary to create a system in which a number of KBS cooperate and combine their problem-solving capabilities. Sometimes this occurs because the problem-solving activity covers a large geographic region (such as in telecommunications networks or military applications), where different KBS have responsibility for different geographical areas; sometimes it occurs because different KBS have different "specialities" to bring to the problem-solving process, similar to the co-operation among human team members. The Multi-Agent System (MAS) architecture has proven a popular and effective method for building a co-operating team of KBS: each KBS in the team is constructed as a software *agent*, conferring abilities of autonomy, self-knowledge, and acquaintance knowledge on the KBS — abilities useful for team-forming and co-operative problem-solving. However, the problem of assessing the reliability of MAS through V&V has received scant attention, which is a matter of some concern because these systems are considerably more complex than individual KBS (Grossner 1995).

The objective of the work described in this paper is to build upon successful techniques in verification of KBS, to develop methods for verifying multi-agent KBS. Specifically, we extend the COVER tool to create COVERAGE: COVER for AGEnts. Section 2 defines the architecture of MAS that we assume, Section 3 describes the properties which COVERAGE is designed to detect, Section 4 describes the COVERAGE tool, and Section 5 concludes. A running example illustrates the use of COVERAGE.

MAS Architecture

Our intention is to provide a general approach to verifying MAS. The architecture of conventional KBS is reasonably well-understood, and the approach taken by the COVER tool is to check KBS represented in a generic *modelling language*, into which many representations can be translated without too much difficulty (Preece, Shinghal, & Batarekh 1992). Using a modelling language rather than the implementation language has three important benefits: the verification approach is not tied to a specific target representation; by abstracting away detail not required for verification, it is computationally easier to work with the modelling language; and not least, verification can be

* This paper is an extended version of the paper "Verification of Multi-Agent Knowledge-Based Systems", presented at the *ECAI-96 Workshop on Validation of KBS.*

† Author's current location: Ericsson Telecommunications, West Sussex, UK.

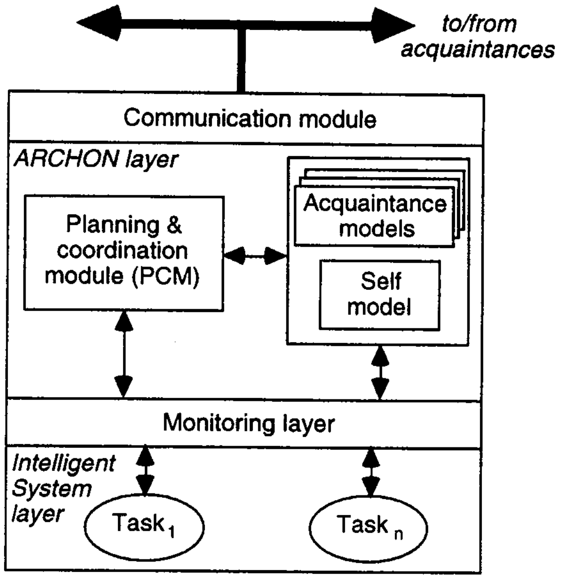

Figure 1: Structure of an ARCHON Agent.

performed prior to implementation (for example, on a conceptual model of the KBS (Wielinga, Schreiber, & Breuker 1992)). These advantages far outweigh the disadvantage, which is that inaccuracies may arise because the modelling language may not exactly match the semantics of the target implementation language — this is fully discussed in (Preece, Shinghal, & Batarekh 1992), where it is shown that such inaccuracies can be dealt with in the verification process.

We would like to take a similar approach to MAS, but their architecture is not as well understood at this time (Wooldridge & Jennings 1995). After careful consideration, the ARCHON[1] architecture (Cockburn & Jennings 1995) was adopted as a starting point for developing a generic representation for COVERAGE, for two main reasons:

- real world systems have been developed within it (Cockburn & Jennings 1995);
- it clearly separates problem-solving KBS domain knowledge from co-operation and communication mechanisms.

The ARCHON approach helps designers correctly decompose and package components of a MAS. The individual components of an ARCHON system are *agents*, which are autonomous and have their own problem solving abilities. Each agent is divided into two separate layers: the *ARCHON Layer* (AL) supports interaction and communication, while the *Intelligent System* layer (IS) provides domain level problem solving — the IS can be a KBS or some other type of information system. This is illustrated in Figure 1.

[1]ARchitecture for Co-operating Heterogeneous ON-line systems.

The agent initiates problem-solving using control rules in the PCM. The PCM rules will typically either invoke co-operation messages to other team members (whose capabilities are defined in the acquaintance models) or by passing a goal to its own IS to solve. The *monitoring layer* between AL and IS acts as a translator between IS and ARCHON terms, and allows the AL to control the IS by invoking goals. For the purposes of this paper we will refer to the knowledge base of the IS layer as *domain knowledge* (DK) and the knowledge base of the AL as the *co-operation knowledge* (CK). Links between CK and DK structures are defined by *monitoring units* (MU).

Example MAS Application

For illustrative purposes, we introduce a MAS in the domain of providing university course advice.[2] Three agents are involved:

User Agent serves only as the user's front-end to the MAS. It has an empty IS layer. Its AL has a user interface which allows the user to enter their identity, year of study, and the name of a course they wish to take. It then locates and contacts an agent who can provide advice on whether a student can take a given course, and relays the agent's response to the user.

Advisor Agent provides advice on whether a student can take a given course. Its IS layer is a course advice expert system; its AL offers the facility for its peers to ask queries of the expert system and obtain answers. To answer queries, the expert system needs

[2]This is a simplified version of a prototype MAS which advises on the options available to students who wish to transfer between universities in the UK.

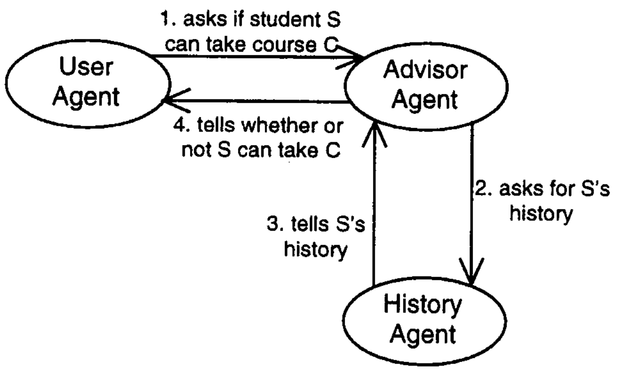

Figure 2: Example MAS Interaction.

information on the academic history of the student (to see if course prerequisites are satisfied); the advisor agent's AL obtains this information by locating and contacting an agent who can provide it.

History Agent provides information on the academic history of students. Its IS layer is a database of student records; its AL offers the facility for its peers to ask queries of the database.

The interaction between the three agents is summarised in Figure 2.

Modelling Language for DK

We are interested in MAS where each agent's DK is a KBS. Therefore, we adopt a slight variant of the COVER KBS modelling language for capturing the DK of an agent. A fuller description of this language, which is built on top of Prolog, appears in (Preece, Shinghal, & Batarekh 1992); here we will summarise the main structures: rules, goals, constraints and data items.

Rules Rules make up most of the DK, defining the flow of problem-solving in either a backward or forward-chaining manner. The following are example COVER rules from the Advisor Agent's IS:

```
rule r42 ::
    canTake(software_eng) if
    knowsC hasValue yes.

rule r43 ::
    knowsC hasValue yes if
    passedCourses includes 'C' or
    passedCourses includes 'C++'.
```

These rules conclude that a student can take the software engineering course if they know the C programming language; they are deemed to know C if they have passed courses on C or C++. Rule consequents conclude a value for some data item, called the rule *subject* (knowsC in rule r43). *Data items* may be boolean facts (canTake(software_eng) in rule r42) or single or multi-valued parameters (knowsC and passedCourses in rule r43).

Goals Goals may be intermediate states in the DK or final conclusions represented within an MU. There will be at least one goal specified for each task required by the ARCHON layer and the MUs will represent one goal each. An example goal declaration is **goal canTake(software_eng)**, where canTake(software_eng) is a data item.

Constraints Constraint declarations permit us to verify the rules. An example declaration from the Advisor Agent's expert system:

```
rule c54 ::
    impermissible if
    canTake(software_eng) and
    canTake(intro_unix).
```

Each declaration states that a combination of data items and their values is semantically invalid: that is, it is not permissible in the real world. The example constraint says that students are not allowed to take both software engineering and introductory UNIX.

Data Declarations Data declarations are used to determine which data items are expected to be supplied as input to the KBS. In a standalone KB this input will be supplied by the user. For an agent the idea is actually the same, except that the "user" is the ARCHON layer, and data declarations now define which data the ARCHON layer has to give the IS in order to achieve a task. An example declaration:[3]

```
data passedCourses / set /
    ['C', 'C++', intro_unix,
     software_eng, ...].
```

The data declarations are used in verification to establish the types of data items, to determine possible legal values that data items can take, and to check that certain values for data items can be established. This example lists possible values for the multi-valued data item passedCourses (not all values are shown).

[3]In COVER, input data are defined askable; here we use the more general keyword data.

Modelling Language for CK

The following language is derived from the description of components in the ARCHON layer (Cockburn & Jennings 1995), consisting of PCM rules to govern the behaviour of the agent, and a set of declarations that define internal data for the agent (beliefs, goals, domain facts) and capabilities available internally (MUs, self model) and externally (aquaintance models).

PCM Rules The PCM rule declarations define the behaviour of the agent. For our purposes here, PCM rules represent only the communication of information and the completion of requested tasks. The following rule is used by the Advisor Agent to determine if a student can take a given course in a given year.

```
pcm_rule p3 ::
    canTakeCourse hasValue yes if
    student hasValue S and
    year hasValue Y and
    course hasValue C and
    agentModel(HistAgent, getHistory,
        [student, year], [history]) and
    request(HistAgent, getHistory,
        [student, year]) and
    activate(checkPrereqs,
        [course, history]) and
    hasPrereqs(C) hasValue yes.
```

This rule uses the `request` operator to find information from an acquaintance;[4] the `activate` operator is used to invoke a task within an agent's own IS. The student S can take the course C if there is an agent `HistAgent` that can deliver S's history, and the task `checkPrereqs` performed by the Advisor Agent's IS yields the fact that `hasPrereqs(C) hasValue yes`.

Agent and Self Declarations Each acquaintance known to an agent has one or more entries in the agent's acquaintance model table of the form:

```
agentModel(Name, Task, Inputs, Outputs).
```

In the previous example PCM rule p3, the Advisor Agent refers to an entry in its acquaintance model, `agentModel` to find the identity of another agent (`HistAgent`, a Prolog variable) which can perform task `getHistory` using data `student` and `year` to produce a value for `history`.

Self models are very similar: they are a self-reference model for the agent's own capabilities, and refer to MU items.

[4]The `request` operator waits for a response as in a conventional remote procedure call; it could be implemented using an agent communication language such as KQML (Finin *et al.* 1994), but in this case the agent would have to wait explicitly for an appropriate reply from its peer.

Monitoring Units The MUs provide a bridge between AL task terms and IS goal terms, allowing for different naming in each (this is necessary to allow existing KBS to be "wrapped" as agents). An example MU for the Advisor Agent's `checkPrereqs` task:

```
mu(checkPrereqs,
   [mapping(course, module),
    mapping(history, passedCourses)],
   [mapping(hasPrereqs, canTake)]).
```

Here, the AL operates in terms of input data `course` and `history`, while the equivalent terms at the IS layer are `module` and `passedCourses`. The IS output term `canTake` (seen in IS rules r42 and r43) corresponds to the AL term `hasPrereqs` (seen in rule p3).

Example Agent Interaction

The User Agent's AL contains the following rule fragment:

```
...
% read values for student, year, course
...
agentModel(AdvisorAgent, courseAdvice,
    [student, year, course],
    [canTakeCourse]) and
request(AdvisorAgent, courseAdvice,
    [student, year]) and
...
% print value of canTakeCourse
...
```

The User Agent locates a peer who can perform the `courseAdvice` task, given input values for `[student, year, course]`, and returning a value for `canTakeCourse`; in this way, it will find the Advisor Agent, and invoke the Advisor Agent's task involving PCM rule p3 shown earlier. Rule p3 interacts with History Agent as shown in steps 2 and 3 in Figure 2, before `activat`ing its own IS task `checkPrereqs` to obtain a value for `hasPrereqs(C)`. Once the term mappings defined in the MU for `checkPrereqs` have been applied, the expert system will attempt to establish a value for `hasPrereqs(C)` (using IS rules such as r42 and r43). If the value is `yes`, rule p3 succeeds and Advisor Agent can return a positive response to User Agent.

MAS Anomalies

In defining the anomalies that can exist in a MAS, modelled using the DK and CK languages described above, we identify four distinct types:

DK anomalies Anomalies confined to an agent's DK.

CK anomalies Anomalies confined to an agent's CK.

CK-DK anomalies Anomalies in the interaction between an agent's CK and DK (that is, between the modelled AL and IS).

CK-CK anomalies Anomalies in the interaction between acquaintance agents (this is, between different agents' ALs).

The first three types are also known as *intra-agent anomalies*; CK-CK anomalies are also known as *inter-agent anomalies.*

DK Anomalies

Essentially, the DK is a KBS; therefore, the COVER anomalies are applicable to the DK. COVER defines four types of anomaly (Preece, Shinghal, & Batarekh 1992); these, and their effects in MAS are as follows:

Redundancy Redundancy occurs when a KBS contains components which can be removed without effecting any of the behaviour of the system. This includes logically subsumed rules (`if p and q then r`, `if p then r`) rules which cannot be fired in any real situation, and rules which do not infer any usable conclusion. In MAS, simply redundant DK components will not affect an agent's ability to achieve goals, but it can lead to inefficient problem-solving and maintenance difficulties (where one rule is updated and its duplicate is left untouched). However, redundancy is usually a symptom of a genuine fault, such as where an unfirable rule or an unusable consequent indicates that some task is not fully implemented. For example, rule r43 in the earlier example would not be firable if the data declaration for `passedCourses` did not contain either of the values `'C'` or `'C++'`.

Conflict Conflict occurs when it is possible to derive incompatible information from valid input. Conflicting rules (`if p then q`, `if p then not q`) are the most typical case of conflict. Conflicting DK is as harmful for an agent as conflicting knowledge in a standalone KBS: the agent can derive incompatible goals from a valid set of input to a task. In human terms, it is untrustworthy. This would be the case in the example if Advisor Agent's IS contained the following rule:

```
rule r44 ::
    canTake(intro_unix) if
    knowsC hasValue yes.
```

The combination of rules r42 and r44 firing would violate constraint c54.

Circularity Circularity occurs when a chain of inference in a KB forms a cycle (`if p then q`, `if q then p`). It may then not be possible for the KBS or agent's DK to solve some goals.

Deficiency Deficiency occurs when there are valid inputs to the KB for which no rules apply (p is a valid input but there is no rule with p in the antecedent). If an agent's DK is deficient, then it may be promising a capability that it cannot supply: if some `agentModel` in an agent's AL indicates that the agent can solve some goal using some input, and the DK is deficient for some combination of that input, it is effectively in "breach of contract". We expand on this theme below.

CK Anomalies

Anomalies confined to an agent's CK fall into two types:

- faulty integrity between components, checkable by cross-referencing PCM rules, agent models, and MUs;
- anomalies in PCM rules: as in DK, PCM rules may be redundant, conflicting, circular and deficient.

CK-DK Anomalies

Anomalies indicating faulty integrity between an agent's CK and DK will typically result in a "breach of contract": the agent's AL is effectively promising that the agent has certain capabilities, and the agent's IS is meant to provide those cababilities. If there is a mis-match between CK and DK, then a promise made by the AL will not be honoured by the IS. If an agent A has a CK-DK anomaly, this may cause a problem for A itself, if it relies on being able to fulfill a task in its self-model; or it may cause a problem for an acquaintance of A, say B, even where B's `agentModel` of A is consistent with A's self-model.[5]

CK-DK anomalies may be simple cases of malformed MUs, where the MU does not properly map to DK terms, or they may be much more subtle cases of unachievable tasks, where the information defined in the MU does not permit the required goal(s) to be achieved by the IS under any possible chain of reasoning. For example, if the MU for task `checkPrereqs` failed to map `history` into `passedCourses`, then the task would be unachievable because pre-requisite checking rules such as r43 could not fire.

CK-CK Anomalies

Again, these anomalies result in "breach of contract" between agents: typically, the `agentModel` held by agent A of agent B is is in conflict with B's self-model. Special cases include:

- *Over/under-informed task:* the task input of the self-model declaration is a subset/superset of its counterpart in the agent model declaration; for example, the User Agent's model of the Advisor Agent's task `courseAdvice` would be under-informed if it neglected to include `year` in its input.
- *Insufficient output:* the task output of the self-model declaration is a subset of its counterpart in the agent model declaration; for example, if the User Agent's model of the Advisor Agent included the additional

[5]If B's `agentModel` of A is not consistent with A's self-model, then this is a CK-CK anomaly, discussed later.

output `prereqs` from the `courseAdvice` task (to indicate which pre-requisites are needed to take the course), then the Advisor Agent's actual capability to perform `courseAdvice` would be insufficient to meet the User Agent's need.

- *Unobtainable item:* an item in an input list to a task in agent A does not appear in any output lists for acquaintance models held by A. This would be the case if the Advisor Agent did not know any peers who could supply a value for `history`.

In comparing the CK rules of different agents, we are assuming that they share a common terminology — they commit to a common *ontology*. Verifying an agent's commitment to an ontology is outwith the scope of this paper; it is the subject of a companion work (Waterson & Preece 1997).

Coverage

The COVERAGE tool has been implemented to detect the anomalies described above in MAS modelled by our DK and CK languages. COVERAGE includes and extends COVER. Each agent in the MAS is presented to COVERAGE in two files, for example `advisor.dk` and `advisor.ck`. The procedure followed by COVERAGE is as follows:

1. COVERAGE invokes COVER to check the `.dk` file in its normal manner, producing a set of reference tables and reporting instances of its four types of anomaly (redundancy, conflict, circularity and deficiency). This covers all DK anomalies.
2. COVERAGE checks the `.ck` file to detect all CK anomalies and build CK reference tables.
3. COVERAGE then takes the reference tables produced by COVER, and cross-references these with the CK reference tables for the agent, reporting CK-DK anomalies.
4. Once the above procedure has been applied for every agent in the MAS, COVERAGE cross-checks all CK reference tables for CK-CK anomalies.

COVERAGE is implemented mostly in Prolog (except for part of COVER which was implemented in C). Indexing of the reference tables makes the above checking operations reasonably efficient for a moderately-sized MAS (complexity of the CK-CK checks is obviously proportional to the sizes of the `agentModel` tables held by the acquaintances). The most expensive check is the CK-DK check that ensures that tasks defined in MUs can be carried out by the IS: this involves checking the environment table created by COVER, containing in effect every inference chain in the IS (Preece, Shinghal, & Batarekh 1992). Arguments in support of COVER's tractability apply here: IS in practice tend not to have very deep reasoning, leading to short inference chains which are exhaustively computable.[6]

[6]Where this is not the case, heuristics as in SACCO can be used to constrain the search for anomalies (Ayel & Vignollet 1993).

Conclusion

COVERAGE is merely a first step — albeit a necessary one — towards the verification of MAS. One difficulty at the outset was defining a stable architecture for MAS, given the current immaturity of the technology. We elected to base our CK and DK languages on the ARCHON framework since ARCHON has been proven to be "industrial-strength", and the notion of agents-as-KBS is well-supported by ARCHON. Moreover, the ARCHON framework introduced the important issue of separating the agent's components into distinct layers, allowing COVER to be used without modification to verify the KBS at the heart of each agent. It can only be hoped that our approach will be applicable to a wider class of MAS architectures.

The following points summarise the current status of our work:

- generic modelling languages have been defined for agent domain and co-operation knowledge;
- a four-layer framework of anomalies has been defined for MAS;
- COVER has been applied successfully to verify agent domain knowledge;
- a new tool, COVERAGE has been built to detect all four layers of MAS anomaly.

Future work in the COVERAGE project includes the following areas.

Implementation of further CK-DK anomalies The anomalies defined for CK-DK "breaches of contract" are largely sub-anomalies of a broader "unachievable task" anomaly. There are other broader anomalies that we have identified but not implemented in COVERAGE, including:

- *Redundancy*, where an agents' capabilities are subsumed by those of other agents: in the extreme case, an agent can be removed from a team without affective the team's abilities as a whole (the agent is literally "made redundant"); less extreme is the issue of redundant tasks performed by an agent.
- *Circularity*, where two or more tasks may become deadlocked.
- *Deficiency*, where agents fail in general to deal with their environment; this is probably the most difficult general anomaly to verify for MAS.

Dynamic anomalies In addition to the verification of statically-detectable anomalies, it is necessary to verify dynamic MAS behaviour. For example, an agent's mechanism for planning and re-planning task achievement, coupled with communication and interaction, is extremely complex to test. It is difficult to envision a dynamic MAS verification system that is omniscient, because the space of all possible interactions will usually be vast or infinite. A possible method of circumventing the problem would be to make the verification system an agent in itself, and to let it verify the process as it is happening.

Real world systems The work carried out so far has demonstrated the use of COVERAGE on a simple prototype MAS, showning that the anomalies discussed are important. Future work is planned which will evaluate the COVERAGE approach on real world MAS applications.

References

Ayel, M., and Vignollet, L. 1993. SYCOJET and SACCO, two tools for verifying expert systems. *International Journal of Expert Systems: Research and Applications* 6(2):357–382.

Cockburn, D., and Jennings, N. 1995. ARCHON: A distributed artficial intelligence system for industrial applications. In *Foundations of Distributed Artificial Intelligence.* New York: John Wiley & Sons.

Finin, T.; Fritzson, R.; McKay, D.; and McEntire, R. 1994. KQML as an Agent Communication Language. In *Proceedings of Third International Conference on Informatio n and Knowledge Management (CIKM'94).* ACM Press.

Grossner, C. 1995. *Models and Tools for Co-Operating Rule-Based Systems.* Ph.D. Dissertation, Concordia University.

Gupta, U. G. 1990. *Validating and Verifying Knowledge-based Systems.* Los Alamitos, CA: IEEE Press.

Preece, A. D.; Shinghal, R.; and Batarekh, A. 1992. Principles and practice in verifying rule-based systems. *Knowledge Engineering Review* 7(2):115–141.

Waterson, A., and Preece, A. 1997. Knowledge Reuse and Knowledge Validation. Submitted to *AAAI-97 Workshop on Verification and Validation of Knowledge-Based Systems.*

Wielinga, B. J.; Schreiber, A. T.; and Breuker, J. A. 1992. KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition* 4(1):5–54.

Wooldridge, M., and Jennings, N. R. 1995. Intelligent agents: Theory and practice. *Knowledge Engineering Review.* Forthcoming.