From Abstract Qualities to Concrete Specification using Guidance Policies*

(Extended Abstract)

Scott J. Harmon
Kansas State University
234 Nichols Hall
Manhattan, Kansas 66506
harmon@ksu.edu

Scott A. DeLoach
Kansas State University
234 Nichols Hall
Manhattan, Kansas 66506
sdeloach@ksu.edu

Robby
Kansas State University
234 Nichols Hall
Manhattan, Kansas 66506
robby@ksu.edu

ABSTRACT

This research describes a formal method for transforming abstract qualities into concrete specification in the form of policies within a multiagent system design methodology.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Specification techniques*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Design, Perfomance, Reliability, Theory.

Keywords

Agent Oriented Software Engineering, Multiagent Systems, O-MaSE, Policy, Law, guidance.

1. INTRODUCTION

Organization-based multiagent systems engineering has been proposed as a way to design complex adaptable systems [1]. Agents interact and can be given goals depending on their individual capabilities. The system designer is faced with the task of designing a system to not only meet functional requirements, but also non-functional requirements. We view these non-functional requirements as abstract qualities of the system. The system designer requires tools to help them generate specifications and evaluate design decisions early in the design process. The approach we are taking is to first generate a set of system traces (possible execution paths) using the models generated at design time. Second, we analyze the system traces using additional information provided by the system designer. Third, we generate a set of policies that will guide the system toward certain abstract qualities.

2. QUALITY METRICS

ISO 9126 [2] defines a set of qualities for the evaluation of software. This document breaks down software qualities into six general areas: functionality, reliability, usability, efficiency, maintainability, and portability. These characteristics are broad and may apply in different ways to different types of systems. In our research, we identified a group of metrics that evaluate multiagent system traces in order to illustrate our concepts. Each metric is formally defined and may be measured precisely over the current system design.

2.1 System Traces

The set of system traces form a tree. We use this structure to determine what choices to make at each decision point in order to maximize (or minimize) some metric. The choice may be what role or what agent to use to achieve a goal, or even what goal to pursue (in the case of OR goals). The divergence in the traces may also happen due to changes in goal parameters, which are normally beyond the control of the system. Thus, the metrics and policies generated may need to take into account some aspect of the goal parameter.

We used Bogor [3] to generate the system traces using the models defined by the the Organization Model for Adaptive Computational Systems (OMACS) [1] meta-model. Bogor is an extensible, state of the art model checker. The traces consist of a sequence of *agent goal assignment* achievements. We generate only traces in which the top-level goal is achieved (system success).

2.2 Efficiency

We are using a trace-length based definition of efficiency. The shorter the trace the more efficient is the top-level goal achievement. Our strategy here is to make assignments such that we minimize the total expected trace length. This minimization will be dynamic in that it will take into consideration the current trace progress. Thus, we consider goal achievements the system has made when determining shortest trace length to pursue. We then convert this logic statically into a set of policies that when enforced will exhibit the same behavior as a minimization algorithm, given current system goal achievements. Given the initial system state, we prefer to make assignments to achieve the overall shortest path. Thus, we will generate directing policies proscribing

*(Produces the permission block, copyright information and page numbering). For use with ACM_PROC_ARTICLE-SP.CLS V2.6SP. Supported by ACM.

Cite as: From Abstract Qualities to Concrete Specification using Guidance Policie, (Extended Abstract), Scott J. Harmon, Scott A. DeLoach, Robby, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 1343–1344

assignments (they may still have multiple options).

Expected trace length is defined in Formula 1.

$$\text{ExpLength}(\xi) = \sum_{i=1}^{n} \frac{1}{1 - pf_i} \quad (1)$$

ξ represents a single trace, while pf_i is the probability of failure for the assignment achievement i within that trace. An *assignment achievement* is the accomplishment of a goal that was assigned to an agent. Thus, an *assignment achievement failure* is a failure of the agent to complete its assignment. The assignment achievement failure probability is defined in terms of capability failure given the assignment. The probability of failure for the agent goal assignment achievement (pf_i) is the maximum of the probability of failure for all the capabilities required for the role in the assignment:

$$pf_i = \max_{c_x \in \text{capreq}(As_i)} Pf(c_x, As_i) \quad (2)$$

It is evident here that some traces will have an infinite expected length (when the failure probability is 100%).

The failure probabilities may be represented as a matrix. For a given capability, only certain portions of the assignment may be relevant to the probability of failure. In these cases, the matrix may be collapsed to a more compact representation.

2.3 Quality of Product

Quality of achieved goals falls under the ISO software quality of Functionality. Here we define quality as a measure of the quality of goal achievement. Quality of goal achievement may depend on the Role, Agent, and Goal Instance (including parameters). In our analysis, we limit ourselves to these influences although goal achievement quality may also depend on such things as environment, history of the system, and current assignments.

Certain roles may obtain a better result when achieving certain goals, likewise certain agents may play certain roles better than other agents. These properties can be known at design time. Usually this intuition is known by the implementers and they may manually design an agent goal assignment algorithm to favor these assignments.

For our analysis and experiments, we mapped assignments to scores. The higher the score, the higher the quality of product. The scores can be specified over the entire assignment, or just portions. For example, role R_1 may achieve goal G_1 better than role R_2 when the parameter of G_1 is of class x. A designer could also specify agents who produce higher quality products and thus may be part of the score determination.

To analyze an entire trace, we score the trace by computing the average quality of product. We realize some products may be more important than others. Without loss of generality, however, we use a simple average as given in Formula 3 (the analysis could potentially include a more-important relationship between products). Let ξ_i be the ith agent goal assignment in trace ξ of length n.

$$\text{Qual}(\xi) = \sum_{i=1}^{n} \frac{score(\xi_i)}{n} \quad (3)$$

2.4 Reliability

Sometimes failure should be avoided at all costs, thus, even if there is a probabilistically shorter trace, it could be the case that we choose the longer trace because we want to minimize the chance of any failure. Formally, we want to minimize goal assignment failures. We do this by minimizing the probability of capability failure. Our strategy here is to pick the minimal failure trace given the current completed goals in the system.

We can use the capability failure matrix defined for Efficiency. The score we are trying to minimize is defined as:

$$\text{Fail}(\xi) = \sum_{i=1}^{n} pf_i \quad (4)$$

Where pf_i is defined as in the Efficiency metric (the probability of failure of assignment i within trace ξ).

It is important here to see the distinction between Reliability and Efficiency. Efficiency is concerned with minimizing the expected trace length, while Reliability is concerned with minimizing the total number of failures. Thus, if we are pursuing Efficiency, we may choose a path in which there may be some failures, even through there is a path with no failures, because the expected trace length is shorter in the path with failures. Reliability will always choose the path with less expected failures, even if the path is longer than another.

3. CONCLUSIONS

We have provided a framework for stating, measuring, and evaluating abstract quality requirements against potential multiagent system designs. We do this by generating policies that can be used either as a formal specification, or dynamically within a given multiagent system to guide the system toward the maximization (or minimization) of the abstract quality constraints. These policies are generated offline, using model checking and automated trace analysis.

These policies can be seen to guide the system toward the abstract qualities as defined in the metrics. Our experiments showed significant performance, quality, and reliability increases over a system using the same models, but without the automated policy generation.

4. ACKNOWLEDGMENTS

This research was performed as part of grants provided by the Air Force Office of Scientific Research (FA9550-06-1-0058) and the National Science Foundation (IIS-0347545).

5. REFERENCES

[1] S. A. DeLoach, W. Oyenan, and E. T. Matson. A capabilities based theory of artificial organizations. *Journal of Autonomous Agents and Multiagent Systems*, 2007.

[2] ISO. *ISO/IEC: 9126 Information technology-Software Product Evaluation-Quality characteristics and guidelines for their use.* International Organization for Standardisation (ISO), 1991.

[3] Robby, M. B. Dwyer, and J. Hatcliff. Bogor: An extensible and highly-modular model checking framework. In *Proceedings of the Fourth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 267–276, 2003.