Formal ensemble engineering

J. W. Sanders[1] and Graeme Smith[2]

[1] International Institute for Software Technology
United Nations University, Macao, SAR China
[2] School of Information Technology and Electrical Engineering
The University of Queensland, Australia

Abstract. The 'ensembles' identified by the InterLink working group on Software Intensive Systems comprise vast numbers of components adapting and interacting in complex and even unforseen ways. If the analysis of ensembles is difficult, their synthesis, or engineering, is downright intimidating. We show, following a recent three-level approach to agent-oriented software engineering, that it is possible to specialise that intimidating task to three levels of abstraction (the 'micro', 'macro' and 'meso' levels), each potentially manageable by interesting extensions of standard formal software engineering. The result provides challenges for formal software engineering but opportunites for ensemble engineering.

1 Introduction

Physical ensembles [6] incorporating potentially massive numbers of nodes, which interact with their physical environment and which may be adaptive and intelligent, offer a promising means of building many complex applications. A necessary condition for the widespread adoption and acceptance of physical ensembles, however, is trust in the dependability of such systems. Given the complexity of ensembles, it is our opinion that such trust can be achieved only using formal engineering methods.

The formal methods that exist today have not been developed to handle the complexity and scale proposed for physical ensembles. Hence, there is a need to develop new approaches. The nature of physical ensembles means these new approaches will vary depending on the level of observation applicable to the system being developed. Following Zambonelli and Omicini's summary of agent-oriented software engineering [21], we adopt three levels for observing ensembles: the *micro*, *macro* and *meso* levels (see Figure 1).

The *micro level* is applicable to ensembles which have a manageable number of components. For the purposes of ensemble engineering, we interpret that to mean 'distinct components': there may be a huge number of identical kinds (at this level of abstraction) of each component. Otherwise, engineering would be impractical. At this level, the behaviour of each component and each component interaction can be formally modelled and analysed. An example of such an ensemble is the system of sensors and actuators controlling a smart home designed, for example, for energy efficiency, or *assisted living*, *i.e.*, allowing an elderly or disabled person to live alone.
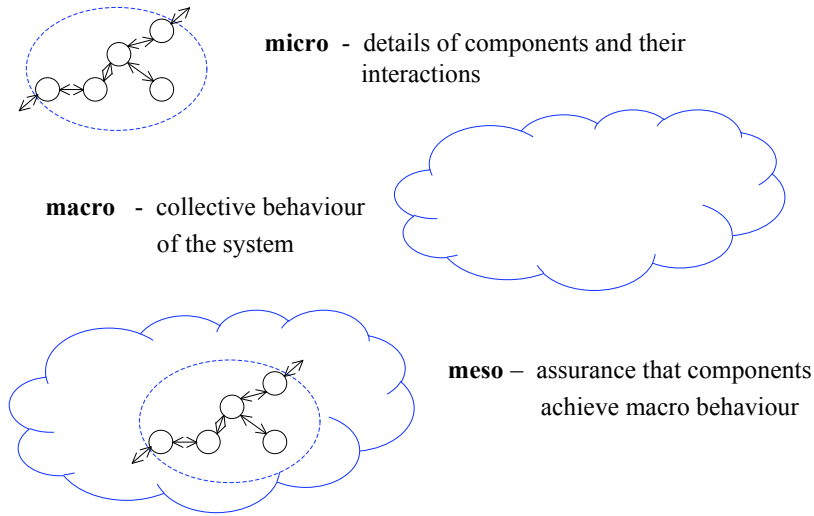
micro - details of components and their interactions

macro - collective behaviour of the system

meso – assurance that components achieve macro behaviour

Fig. 1. Levels of observation

The *macro level* is applicable to ensembles comprising massive numbers of components, possibly distributed over a network and operating in a dynamic and uncontrollable environment. It is not feasible to engineer such systems in terms of individual components. Instead, means of engineering the collective behaviour of the components are required. An example of such an ensemble is an *ad hoc* sensor network deployed in an urban environment, *e.g.*, to monitor traffic jams and accidents and wirelessly communicate such information to drivers in the vicinity [16]. The collective behaviour in this case would be congestion-free traffic flow.

The *meso level* is seen as pertaining when an existing micro-level component is added to an existing macro-level system. In [21], concern centres on verifying that the deployment of a micro-level system within a macro-level one does not compromise the behaviour of either system. In our approach, the macro-level system *is* the specification of the whole system, the micro level contains the implementation, and the meso level embodies designs by which the members of the micro level achieve the behaviour specified at the macro level. For example in the sensor network, the meso level would contain structures to enable (GPS aware) vehicles to communicate with traffic sensors and each other. In top-down development of a system, the meso level bridges the gap between the macro and micro levels, showing what a micro-level component must achieve in addition to its unilateral micro-level behaviour. By requiring that the micro- and macro-level behaviours are not compromised, that returns us to the outlook of [21].

2 Micro-level ensemble engineering

The key challenge at the micro level of observation is the extension and modification of traditional formal methods. Much work has been done on formal methods in the areas of continuous real-time [4, 22], probability [10, 8] and mobility [11, 3], all areas of importance for physical ensembles. However, other areas of equal importance have received only limited attention; in particular, the areas of spatial location, autonomy and intelligence, and adaptation.

2.1 Spatial location

The spatial location of components in some ensembles is vital. For example, in an industrial manufacturing setting the precise location and orientation of a robot working in a team with other robots is a necessary part of its specification if collisions are to be avoided and cooperation achieved. Other applications such as claytronics [5] or free-flight air traffic control [2, 15] also require precise locations of components to be specified.

For some applications a discrete notion of space may be sufficient, in others continuous space may be required. In either case, new formal methods should be developed where space is a first-class concept, rather than just modelled. Approaches to incorporating real-time into formal methods should offer some guidance.

2.2 Autonomy and intelligence

In most existing formal methods, components are reactive. They do not have goals and plans that enable them to act autonomously. Although much work has been done in the artificial intelligence (AI) community on goal-oriented decision making, there has been little integration of this work with formal methods, or with software engineering in general. Most often, AI techniques, when used, are introduced during the implementation phase of a project, rather than during high-level requirements analysis and design phases.

Incorporating AI techniques with formal methods is essential if we are to promote their consideration at the highest levels of system abstraction. Possible approaches include new formal methods based on agent-based approaches [20] and machine learning [13], or on non-standard logics, such as fuzzy logic [7] or non-monotonic logics [1], which can be used to model intelligent decision making processes.

2.3 Adaptation

Components in ensembles will need to adapt their behaviour to respond to unforseen changes in their environment. It is possible to model changes in behaviour within a single specification using, for example, appropriate operators for combining behaviours [19]. It is also possible to do so if the state spaces of the various

adaptations have a uniform abstraction [17]. However, deciding on the kinds of changes which are allowable for truly adaptive components is difficult.

Research on changing high-level requirements of real-time specifications has shown that all such changes can be modelled as a sequence of refinements and a minimal set of basic rules [18]. Similarly, changes to component configurations in an object-oriented setting can be modelled as a sequence of refinements and a minimal set of rules [9]. Hence, it seems feasible that a formal calculus of specification change could be developed.

By establishing a formal relationship between pairs of specifications, such a calculus would enable us to reason about changes to a given specification. In particular, it would enable us to determine the effect a change has on established properties of the specification. This would potentially enable us to reason about the effects of adaptation, and to determine the limits of adaptability that would maintain critical properties at both the component and system levels.

3 Macro-level ensemble engineering

A macro-level system can be *specified* simply as a combination of all the micro-level components (described unilaterally), conjoined with a condition ensuring that the result behaves as desired. Typically that condition captures behaviour that is thought of as being *emergent*: not a consequence of the behaviours of the unilateral micro-level components. Without it, the specification would allow undesirable behaviours resulting from the undisciplined interaction of components at the micro level. Such a specification trades clarity for any hint of implementation strategy. It is at the meso level that the emergent condition is to be achieved, somehow, from the micro components.

In the traffic-sensor example, the micro-level might describe the system in terms of components (cars, public transport and commercial vehicles) 'interacting' on the roads; there are many instances of each component. Then the macro specification would contain those, mediated by a predicate ensuring the smooth flow of traffic. Such behaviour is of course emergent when viewed from the level of an individual vehicle.

It is to be expected that, because of the huge number of components in an ensemble, macro-level behaviour is captured using distributions (in the sense of statistics) and even notions of convergence in space or time (to describe the effects of adaptability in achieving what might be termed 'societal stability'). That must in turn affect the definition of conformance of a design to its specification. But conformance 'at a certain confidence level' may not sit well with abstraction [14] and so must be investigated.

4 Meso-level ensemble engineering

The goal of the meso level is to ensure that refinement holds when a macro-scale specification, *MacroSpecification*, is augmented with a micro-scale component,

MicroComponent, as part of the design process. In terms of the symbol $\sqsubseteq$ for 'valid refinement',

$$MacroSpecification \sqsubseteq MicroComponent \wedge MacroSpecification'.$$

Now the right-hand side specifies a design in which the ingredients combine to achieve the ensemble specification on the left. In the traffic-sensor example, a design achieving free traffic flow might incorporate the relay of information from traffic sensors to vehicles which use GPS and data from neighbouring cars to regulate their velocity (speed in current route or change of route) to avoid traffic jams. It is important to acknowledge the role played by human drivers in the adaptability necessary to achieve emergence: each smart car might offer its driver a choice of possibilities and different driver preferences might be expected to substitute for the randomisation required in network routing algorithms to avoid repeated blockages.

Designs at the meso level are complicated by the fact that components in both the micro and macro levels may be mobile and hence the systems may merge, blurring their boundaries. From the viewpoint of just the micro level, it would be usual to place assumptions on the environment of a component. Similarly, to engineer a macro-level system requires assumptions about the interactions between the components.

To exploit the proposed formal approach, we need to formalise the allowable meso-level interaction patterns and verify that the behaviour and assumptions of the components in the micro-level system conform to these patterns. Suitable formalisms could be built on process algebras, especially those supporting mobility [11, 3], adding elements of, for example, game theory to capture the more complex behaviour possible with autonomous, intelligent components. Also relevant to such formalisms is current work on languages for orchestrating distributed systems [12] at the meso level.

5   Summary

To engineer physical ensembles formally, we propose extensions to traditional formal methods and the way they are applied at three levels of observation:

1. At the *micro level* where we have a manageable number of distinct components, we require extensions to existing formal methods which
   - have a first-class concept of *spatial location*,
   - incorporate AI techniques to capture *autonomy and intelligence*, and
   - have theories, beyond refinement, relating specifications in order to reason about *adaptability*.

2. At the *macro level* where the number of components is massive and it is not feasible to think in terms of individual components, we need a notion of an *emergence condition* which captures the desired collective behaviour of the

system and, importantly, rules out undesirable behaviours. We also need to investigate the conformance of designs to specifications where behaviour is statistically defined.

3. At the *meso level* where we introduce micro-level components as part of designing a macro-scale system, we need formalisms which allow us to capture and reason about complex interaction patterns. It is at this level that we move from the clarity of emergence predicates at the macro level, to the strategies employed to satisfy those predicates at the micro-level.

References

1. G. Brewka, J. Dix, and K. Konolige. *Nonmonotonic Reasoning - An Overview.* CSLI publications, 1997.
2. G. Byrne. Is it time to give airlines the freedom of the skies? *New Scientist*, 2351:12, 2002.
3. L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
4. C.J. Fidge, I.J. Hayes, A.P. Martin, and A.K. Wabenhorst. A set-theoretic model for real-time specification and reasoning. In J. Jeuring, editor, *Mathematics of Program Construction (MPC'98)*, volume 1422 of *Lecture Notes in Computer Science*, pages 188–206. Springer-Verlag, 1998.
5. S.C. Goldstein, J.D. Campbell, and T.C. Mowry. Programmable matter. *IEEE Computer*, 38(6):99–101, 2005.
6. M. Hölzl and M. Wirsing. State of the art for the engineering of software-intensive systems. InterLink Deliverable Number D3.1, 2007. Availabe online at: `http://interlink.ics.forth.gr/central.aspx?sId=84I238I744I323I344283`.
7. G.J. Klir and Bo Yuan. *Fuzzy sets and fuzzy logic: theory and applications.* Prentice Hall, 1995.
8. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Verification of real-time probabilistic systems. In S. Merz and N. Navet, editors, *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, chapter 8, pages 249–288. John Wiley & Sons, 2008.
9. T. McComb and G. Smith. A minimal set of refactoring rules for Object-Z. In *International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2008)*, volume 5051 of *Lecture Notes in Computer Science*, pages 170–184. Springer-Verlag, 2008.
10. A.K. McIver and C.C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Monographs in Computer Science. Springer, 2005.
11. R. Milner. *Communicating and Mobile Systems: The π-Calculus.* Cambridge University Press, 1999.
12. J. Misra and W.R. Cook. Computation orchestration: A basis for wide-area computing. *Software and Systems Modelling*, 6(1):83–110, 2006.
13. T.M. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

14. C.C. Morgan, A.K. McIver, and J.W. Sanders. Probably Hoare? Hoare probably! In J. Davies, A.W. Roscoe, and J.C.P. Woodcock, editors, *Millenial Perspectives in Computer Science*, pages 271–282. Palgrave, 2000.

15. RTCA Task Force 3. Final Report on Free Flight Implementation. RTCA Inc., 1995.

16. M. Saito, J. Tsukamoto, T. Umedu, and T. Higashino. Design and evaluation of inter-vehicle dissemination protocol for propagation of preceding traffic information. *IEEE Transactions on Intelligent Transportation Systems*, 8(3):379–390, 2007.

17. J.W. Sanders and M. Turilli. Dynamics of control. In *Theoretical Aspects of Software Engineering (TASE 2007)*, pages 440–449. IEEE Computer Society, 2007. Expanded version available as: UNU-IIST report 353 at `http://www.iist.unu.edu`.

18. G. Smith. Stepwise development from ideal specifications. In J. Edwards, editor, *Australasian Computer Science Conference (ACSC 2000)*, volume 22 of *Australian Computer Science Communications*, pages 227–233. IEEE Computer Society, 2000.

19. G. Smith. Specifying mode requirements of embedded systems. In M. Oudshoorn, editor, *Australasian Computer Science Conference (ASCS 2002)*, volume 24 of *Australian Computer Science Communications*, pages 251–258. Australian Computer Society, 2002.

20. M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.

21. F. Zambonelli and A. Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.

22. Zhou Chaochen, C.A.R. Hoare, and A.P. Ravn. A calculus of durations. *Information Processing Letters*, 40:269–271, 1991.