Model-Driven Design of Ensemble-Based Component Systems

Ilias Gerostathopoulos

Faculty of Mathematics and Physics
Charles University in Prague
Malostranske Namesti 25, 11800, Prague, Czech Republic
iliasg@d3s.mff.cuni.cz

Abstract. In this research abstract we describe our approach towards the design of ensemble-based component systems. Our motivation lies in the fact that, in these systems, tracing the behavior of individual constituents to system-level goals and requirements is challenging. Our approach is based on a novel invariant-based model that achieves the desired traceability. Along with using the model in a method that allows for systematic contractual design, we employ the model at runtime to achieve dynamic adaptation on the basis of requirements reflection.

Keywords: ensembles, invariants, system design, traceability

1 Introduction

In the beginning, things were not going well. The heavy storm had damaged the network infrastructure so heavily that temperature and moisture sensors on the tarmac could not communicate with their base stations any longer. This meant that continuous analysis of tarmac condition had to stop until the network cables were back in place and sensors started providing fresh measurements to the base stations. In face of the danger of failing in their task to disseminate the sensed data, the sensors switched to ad-hoc communication mode: they propagated their data to software modules inside the vehicles heading towards the base stations; the vehicles acted as network relays for the ad-hoc network and "augmented" sensors for the base stations. Even with considerable delays compared to the default mode, the system managed to keep a sufficient level of operation stability.

Although developing a *software-intensive cyber-physical system* (siCPS) [12] such as the above road sensing system is already technically feasible, there are challenges related to streamlining the design and development of such systems.

DEECo component model [1,4] has been proposed within the ASCENS FP7 project [11] as a modeling approach suitable for the development of siCPS. A DEECo application consists of a number of components and interaction templates, based on which dynamic component groups – *ensembles* [11] – are established at runtime. A DEECo component comprises state (referred to as *knowledge*) and *processes* which periodically read and/or update its knowledge, similar

```
ensemble PropagateTemperatureToVehicles:
 coordinator: TemperatureSensor
 member: Vehicle
 membership:
  distance(coordinator.position, member.position) < THRESHOLD
 exchange:
  member.temperatureMap ←(coordinator.id, coordinator.temperature)
 scheduling: periodic( 15 secs )
```

Fig. 1. Example of a DEECo ensemble definition in the road sensing system.

to processes in a real-time system. Interaction is allowed only between components within an ensemble and takes the form of knowledge exchange. An ensemble definition (Fig. 1) specifies (i) a *membership* condition, i.e., under which condition (evaluated on components' knowledge) one *coordinator* and potentially many *member* components should interact, and (ii) an *exchange* function, i.e., which knowledge exchange should be performed within the established group. We view DEECo as an instantiation of the new class of *ensemble-based component systems* (EBCS), and use it to demonstrate our EBCS design approach.

The **problem** in EBCS is that it is difficult to associate the low-level concepts of periodic computation and conditional knowledge exchange to system-level goals and requirements applicable in different operational contexts. This problem manifests itself both at design time and at runtime. At design time the challenge is: *"How to **design** an ensemble-based system so that its situation-specific system-level goals are consistently mapped to implementation-level artifacts?"*; at runtime the challenge becomes: *"How to **trace** the runtime behavior to situation-specific system-level goals to achieve runtime compliance checking?"*.

The **objective** of this research is thus to investigate the *design dimension* of ECBS and propose a model that provides *dependability* (in the form of traceability to system-level goals) and *adaptability* (in the form of adjusting to different operational contexts/situations). We aim for using the model both to guide the design of EBCS (Sect. 2.1), and to achieve runtime compliance checking and model-based adaptation (Sect. 2.2).

2 Approach: Invariant-Based Model

Our approach is based on the observation that component processes and knowledge exchange activities in EBCS are feedback loops that constantly maintain the property of being within the bounds of normal operation – *operational normalcy*. We have thus proposed the *invariant* concept to model the operational normalcy at every time instant [13]. Syntactically, an invariant is an expression that relates the input and output knowledge of an (abstract) activity, e.g. *"Vehicle's V belief over sensor S::temperature – V::temperatureMap – is updated every 30 secs."*. A key assumption here is that system-level goals in EBCS can also be described declaratively and thus modeled with the invariant construct. For instance, one such high-level invariant in our running example is *"Temperature readings must reach the base stations within 120 secs"*.

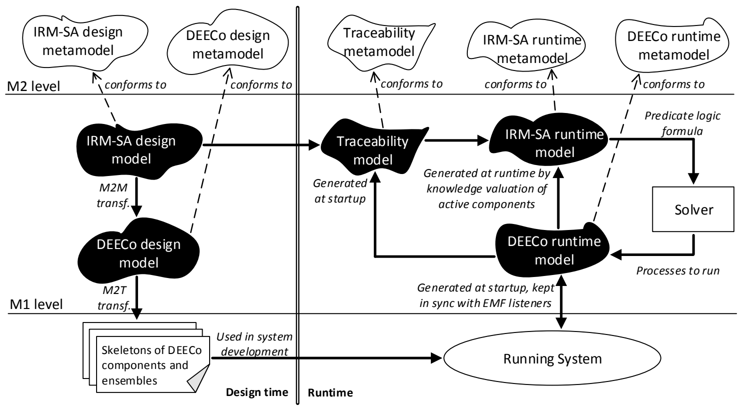

Fig. 2. Overview of the IRM-SA approach.

Armed with the invariant concept, we have proposed the *Invariant-Refinement Method for Self-Adaptation – IRM-SA* [5,13], whose goal is to link high-level invariants (corresponding to system-level goals) to low-level ones (corresponding to concrete activities of the software system). The output of the method is the *IRM-SA design model*; this model can be used (i) to generate DEECo code skeletons via the series of model transformations depicted on the left part of Fig. 2 and (ii) to enable online checking of invariant satisfaction and system adaptation via a models@runtime approach (illustrated on the right part of Fig. 2).

2.1 Design with IRM-SA

In this section we present the design process of IRM-SA. It is a mixed top-down/bottom-up iterative process where invariants are refined into sub-invariants by means of decomposition (e.g. structural elaboration). The process comprises:

1. Identification of the top-level goals and specification of top level invariants of the system-to-be, e.g. invariant [i1] in Fig. 3.
2. Identification of the *design components* by asking "which knowledge does each invariant involve and where is this knowledge obtained from?". At the design stage, a component is a participant/actor of the system-to-be, comprising internal state. In our example, the identified components are the `TemperatureSensor`, `BaseStation`, and `Vehicle`.
3. Decomposition of each non-leaf invariant by asking "**how** can this invariant be satisfied?". Leaf invariants are either *process invariants* (e.g. invariant [p1]) or *exchange invariants* (e.g. invariant [e2]) that can be mapped 1-to-1 to component processes or ensemble definitions, respectively. For instance, the exchange invariant [e2] can be mapped to the `PropagateTemperatureToVehicles` ensemble of Fig. 1.
4. Identification of any other activities that have to be performed in the system and specification of invariants out of them (not demonstrated here).

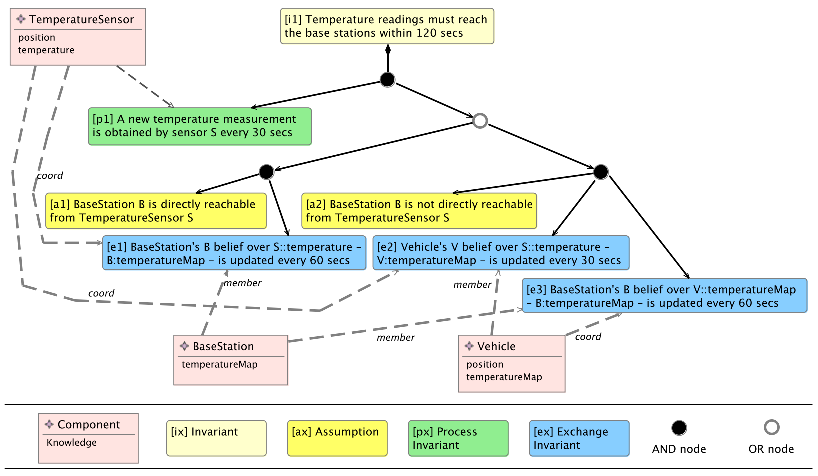
TemperatureSensor
position
temperature
[i1] Temperature readings must reach the base stations within 120 secs
[p1] A new temperature measurement is obtained by sensor S every 30 secs
coord
[a1] BaseStation B is directly reachable from TemperatureSensor S
[a2] BaseStation B is not directly reachable from TemperatureSensor S
[e1] BaseStation's B belief over S::temperature – B:temperatureMap – is updated every 60 secs
[e2] Vehicle's V belief over S::temperature – V:temperatureMap – is updated every 30 secs
[e3] BaseStation's B belief over V::temperatureMap – B:temperatureMap – is updated every 60 secs
member
member
member
coord
coord
BaseStation
temperatureMap
Vehicle
position
temperatureMap
Component
Knowledge
[ix] Invariant
[ax] Assumption
[px] Process Invariant
[ex] Exchange Invariant
AND node
OR node

Fig. 3. Example of an IRM-SA design model.

5. Composition of dangling invariants together by asking "**why** do we need to satisfy these invariants?". This step is also not demonstrated in our example.
6. Capturing of the situation that conditions every *situation-specific invariant* using *assumptions* (e.g. invariant [a1]). An assumption is a special type of invariant that is expected to be maintained by the environment.
7. Identification of alternative (OR) decompositions according to the different situations identified at step 6. In our example, the right-most part of the top-level decomposition is OR-decomposed to capture the fact that different invariants should hold when a `BaseStation` is out of direct reach.

The IRM-SA design process is backed up by a prototype design tool (used to produce the IRM-SA model of Fig. 3) and a Java code generation tool, based on Eclipse's EMF and Epsilon toolchains; both are accessible via `http://d3s.mff.cuni.cz/projects/components_and_services/irm-sa/`.

2.2 Runtime compliance checking and adaptation

To check which invariants hold at runtime and adapt the system accordingly, we follow a models@runtime approach [17]. As a first step, the running system is reflected into an architectural model (*DEECo runtime model* in Fig. 2) that captures the running component processes and established ensembles. Along with a *traceability model*, which contains the mapping between design and runtime artifacts, DEECo runtime model is used to generate another model that captures the runtime state at the requirements level (*IRM-SA runtime model*). This is basically an instantiation of the IRM-SA design model in which design components are mapped to concrete component instances and invariants are associated with boolean values. This is done by associating the invariants and the *computable* assumptions to *monitors* (implemented as Boolean methods in Java) that evaluate the condition under which each invariant is considered to hold.

The second step involves reasoning on the generated IRM-SA runtime model. As an illustration of one possible way to do this, we are translating the model into a predicate logic formula which can be automatically evaluated by a solver (we use Sat4j [16]). The result of the solver is then used to enact changes on the DEECo runtime model (currently by starting/stopping processes corresponding to invariants chosen in the OR-decompositions), which are eventually propagated to the running system, as illustrated on the right-most part of Fig. 2.

A proof-of-concept implementation of IRM-SA-based adaptation is accessible via `http://d3s.mff.cuni.cz/projects/components_and_services/irm-sa/`.

On-going work. We are currently investigating (i) the fuzzification of invariant evaluation to achieve more fine-grained control, and (ii) more elaborate adaptation actions (e.g. changing a component's period at runtime). To evaluate our approach we are conducting experiments to measure the applicability of our adaptation loop in practical settings (e.g. in face of frequent component disconnections). We have also designed and conducted a pilot of a controlled experiment (empirical study) to evaluate the effectiveness of the IRM-SA process.

3 Related Work

Systematic elaboration of requirements has been advocated by goal-oriented requirements engineering approaches, such as KAOS [7,15] and Tropos [3,9]. Although we draw inspiration from them we differentiate in the following [8]: (i) neither KAOS nor Tropos are tailored for ensemble-based systems, whereas IRM-SA provides a direct translation to the implementation-level concepts of autonomous components and ensembles; (ii) compared to KAOS, the objective of the IRM-SA method is not to create requirements documents (e.g., SRS), but software architectures; (iii) compared to Tropos, which also supports design of dynamic systems, IRM-SA concepts (i.e., invariants) do not focus on future states (like goals in Tropos), but on knowledge evaluation at every time instant, fitting better the design of feedback loop-based systems.

Our approach towards adaptation fits into the conceptual model proposed by Kramer and Magee [14], where the IRM-SA model stands as a domain-specific goal management layer. Our adaptation mechanism also follows the proposals for explicit representation of requirements as runtime entities [2,6].

Compositional definition of architecture configurations based on individual variation points and runtime reconfiguration is also employed in Dynamic Software Product Lines [10]. Our main difference is that, in IRM-SA, decomposition carries the formal semantics of refinement, i.e., in an AND (resp. OR) decomposition the conjunction (resp. disjunction) of the children entails the parent.

Acknowledgments. The research leading to these results has received funding from the European Union Seventh Framework Programme FP7-PEOPLE-2010-ITN under grant agreement n°264840.

References

1. Al Ali, R., Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F.: DEECo: An Ecosystem for Cyber-Physical Systems. In: Companion Proc. of ICSE'14, Hyderabad, India. pp. 610–611. ACM (Jun 2014)
2. Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements Reflection: Requirements as Runtime Entities. In: Proc. of ICSE '10, Cape Town, South Africa. pp. 199–202. ACM (2010)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (May 2004)
4. Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F.: DEECo – an Ensemble-Based Component System. In: Proc. of CBSE'13, Vancouver, Canada. pp. 81–90. ACM (Jun 2013)
5. Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F., Plouzeau, N.: Adaptation in Cyber-Physical Systems: from System Goals to Architecture Configurations. Tech. rep., D3S-TR-2014-01, Charles University (Jan 2014), `http://d3s.mff.cuni.cz/publications/download/D3S-TR-2014-01.pdf`
6. Cheng, B., et al.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems, LNCS, vol. 5525, pp. 1–26. Springer Berlin Heidelberg (2009)
7. Dardenne, A., Van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(April), 3–50 (1993)
8. Gerostathopoulos, I., Bures, T., Hnetynka, P.: Position Paper: Towards a Requirements-Driven Design of Ensemble-Based Component Systems. In: Proc. of HotTopiCS workshop of ICPE'13. pp. 79–86. ACM (Apr 2013)
9. Giorgini, P., Kolp, M., Mylopoulos, J., Pistore, M.: The Tropos Methodology: An Overview. In: Methodologies and Software Engineering for Agent Systems, pp. 89–106. Kluwer Academic Publishers (2004)
10. Hinchey, M., Park, S., Schmid, K.: Building Dynamic Software Product Lines. Computer 45(10), 22–26 (Oct 2012)
11. Hölzl, M., et al.: Engineering Ensembles: A White Paper of the ASCENS Project. ASCENS Deliverable JD1.1 (2011), Online: `http://www.ascens-ist.eu/whitepapers`
12. Hölzl, M., Rauschmayer, A., Wirsing, M.: Engineering of Software-Intensive Systems: State of the Art and Research Challenges. In: Wirsing, M., Banâtre, J.P., Hölzl, M., Rauschmayer, A. (eds.) Software-Intensive Systems and New Computing Paradigms, LNCS, vol. 5380, pp. 1–44. Springer Berlin Heidelberg (2008)
13. Keznikl, J., Bures, T., Plasil, F., Gerostathopoulos, I., Hnetynka, P., Hoch, N.: Design of Ensemble-Based Component Systems by Invariant Refinement. In: Proc. of CBSE'13, Vancouver, Canada. pp. 91–100. ACM (Jun 2013)
14. Kramer, J., Magee, J.: A Rigorous Architectural Approach to Adaptive Software Engineering. Journal of Computer Science and Technology 24(2), 183–188 (2009)
15. Lamsweerde, A.V., Darimont, R., Letier, E.: Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Trans. on Soft. Engin. 24(11), 908–926 (1998)
16. Le Berre, D., Parrain, A.: The Sat4j library, release 2.2. Boolean Modeling and Computation 7, 59–64 (2010)
17. Morin, B., Barais, O., Jezequel, J.M., Fleurey, F., Solberg, A.: Models at Runtime to Support Dynamic Adaptation. Computer 42(10), 44–51 (2009)