Requirements-Driven Adaptation: Compliance, Context, Uncertainty, and Systems

Amit K. Chopra
University of Trento, Italy
chopra@disi.unitn.it

***Abstract*—The systematic study of software self-adaptation has emerged as one of the key areas of software engineering. The challenges and the ontology relevant to this area are still being formulated. I take this opportunity to present some of my observations on compliance, context, and uncertainty as they pertain to adaptation. I also argue that requirements engineering, and to a large extent software engineering, takes a centralized perspective of systems, and therefore cannot model let alone enable reasoning about adaptation in multiagent systems.**

***Index Terms*—Verifiability, Crisp requirements, Fuzzy requirements, Critical requirements, Multiagent Systems, Interaction, Commitments, Requirements engines**

I. Introduction

In recent years, software self-adaptation has emerged as one of the focus areas in software engineering. Self-adaptation is not a new goal in itself; in fact, it has been applied in systems design all along. The Internet was designed to adapt to routing failures and the discovery of new routers. Self-tuning databases could decide which indexes to create thereby making database administration simpler. Work in multiagent teamwork has considered adaptation via the formation of joint intentions and plans. What we are witnessing now is the push to understand adaptation systematically in terms of software requirements and architecture so that we may design systems with adaptation in mind.

Requirements modeling approaches deal in such high-level concepts as stakeholder goals, capabilities, responsibilities, and goal dependencies. Lately, various manifestos and broad technical outlines have been published that advocate systems reflecting upon their own requirements models at runtime in order to effect adaptation [1], [2]. In such systems, the requirements are usually represented in terms of goals. The basic idea is that a system would monitor the satisfaction of its own goals and if some goals fail, then the system would adopt alternative goals. Architectural approaches typically assume an adaptation manager that rewires together components based upon observed events [3]. In a sense, RE approaches address adaptation at the strategic level: ultimately, even architectural adaptation would be driven by the necessity to fulfill high-level requirements [4]. The advantage of effecting adaptation based upon high-level models, whether of requirements or architecture, is that technical details such as the exact plans and procedures may be left to the infrastructure.

I take this opportunity to put forward some observations about requirements-driven adaptive systems. I take the following positions.

- Compliance is key. Requirements must be *crisp* in that one should be able to check if systems comply with them. Some new requirements languages fail this criterion.
- Modeling context is key. The distinction between critical (invariant or crisp) and noncritical (fuzzy) requirements is misleading. Instead, it is much better to think of requirements as being contextual.
- Adaptation to unforeseen circumstances is not realistic. It is better to pay more attention to the methodological details of requirements specification so that we have robust specifications.
- Modeling interaction and social commitments among agents is key to adaptation in multiagent systems. Currently, RE lacks the abstractions to model such systems.

I discuss each of these positions one by one and then conclude the paper with some additional observations.

II. Compliance

The key property that ties systems with their requirements is that of *compliance*: one should be able to tell if a system is compliant with, in other words, meets its requirements. In general, stakeholders want systems that are compliant with requirements. However, it is well known that determining software compliance by testing and verification is not easy, especially in open settings. In general, we can only talk about our confidence that the system meets its requirements. That is the reason why mission critical systems often have redundancy built into them—redundancy is a way to increase confidence.

However, what we should not compromise upon is the ability to tell whether a system is compliant or not by observing its operation. Thus during operation, if the rate of flow of coolant in a nuclear reactor fails to meet requirement, we should be able to detect that. Detecting noncompliance is crucial to our ability improving systems design. Once noncompliance is flagged, we can analyze the chain of events that led to noncompliance and take preventive measures.

Detecting noncompliance at runtime necessarily implies that the requirements be crisp. The requirement $\mathbf{R_0}$: *the rate of coolant flow shall be high* does not meet this criterion because we do not know what "high" means. $\mathbf{R_1}$: *the rate of coolant flow shall not drop below 3000 gallons per second* is crisp though. (Requirements such as $\mathbf{R_0}$ are often characterized as *soft goals* in the literature. $\mathbf{R_1}$ may be interpreted as a metricized version of $\mathbf{R_0}$). $\mathbf{R_2}$: *the rate of coolant flow shall remain in the range of 2800-3200 gallons per second*

Label	Requirement	Crisp
$\mathbf{R_0}$	The rate of coolant flow shall be high	No
$\mathbf{R_1}$	The rate of coolant flow shall not drop below 3000 gallons per second	Yes
$\mathbf{R_2}$	The rate of coolant flow shall remain in the range of 2800-3200 gallons per sec	Yes
$\mathbf{R_3}$	The backup coolant pump shall be operational 98% of the time	No
$\mathbf{R_4}$	The mean time between failure for the coolant pump shall be more than two years	No
$\mathbf{R_5}$	The pump shall not malfunction more than once in any 730 day period	Yes
$\mathbf{W_0}$	The fridge shall detect and communicate with all food packages	Yes
$\mathbf{W_1}$	The fridge shall detect and communicate with AS MANY food packages AS POSSIBLE	No
$\mathbf{W_2}$	The system shall raise an alarm if no activity by Mary is detected for some hours (to be decided) during normal waking hours	Yes
$\mathbf{W_3}$	...EVENTUALLY, all devices SHALL use the same data	No

TABLE I
SOME EXAMPLES OF CRISP AND NONCRISP REQUIREMENTS. CRISP REQUIREMENTS ARE THOSE THAT SUPPORT VERIFYING COMPLIANCE AT RUNTIME

is likewise crisp. $\mathbf{R_3}$: *the backup coolant pump shall be operational 98% of the time* is not crisp because the number of observations are unbounded. For the same reason $\mathbf{R_4}$: *the mean time between failures for the coolant pump shall be more than two years* is not a crisp requirement. $\mathbf{R_5}$: *the pump shall not malfunction more than once in any 730 day period* is crisp though. Table I requirements summarizes this discussion. In summary, *a requirement is crisp if and only if compliance with it can be verified at runtime*.

Whittle et al. [5] introduce a language RELAX with new operators for the express purpose of supporting adaptation under uncertainty. They consider two kinds of requirements: requirements that are invariants and requirements that are not. Invariant requirements are understood to be critical, whereas the others are understood to be noncritical. The idea is that whereas a system must meet the critical requirements, the system has leeway, that is, room for adaptation, in meeting the noncritical requirements. Whittle et al. consider a smart home setting. According to them, the requirement $\mathbf{W_0}$: *a fridge shall detect and communicate with all food packages* is invariant. By contrast, Whittle et al. do not deem the requirement $\mathbf{W_1}$: *the fridge shall detect and communicate with AS MANY food packages AS POSSIBLE* an invariant; they deem it *relaxed*. Whittle et al. consider another requirement $\mathbf{W_2}$: *the system shall raise an alarm if no activity by Mary is detected for some hours (to be decided) during normal waking hours*. They deem $\mathbf{W_2}$ an invariant. The advantage Whittle et al. claim is that if the requirements specification is $\{\mathbf{W_1}, \mathbf{W_2}\}$, then in situations where all available resources are required to satisfy $\mathbf{W_2}$, the system could forgo the satisfaction of $\mathbf{W_1}$. Besides quantitative RELAXed requirements expressed using operators such as AS MANY AS POSSIBLE, one can also express temporal RELAXed requirements using operators such as AS EARLY AS POSSIBLE, AS LATE AS POSSIBLE, and so on. Baresi et al. [6] introduce a language with operators similar to RELAX to support runtime adaptation. We refer to such operators as *relaxed operators* and requirements expressed using relaxed operators as *relaxed requirements*.

Relaxed requirements are not crisp. In other words, such requirements do not yield to compliance checking. Even a system where the fridge was designed to *never* monitor its contents could not be proven noncompliant with $\mathbf{W_1}$ by observing its operation. The reason is the unbounded nature of these requirements: it places no time or quantity bounds.

Requirements specifications are necessarily prescriptive. There avails no benefit from considering requirements as noncritical in the sense of Whittle et al. (equivalently, fuzzy in the sense of Baresi et al.). The users of a system may view some requirement as being uncritical but that just means we are willing to tolerate noncompliance with that requirement. That does not mean a requirement cannot be flexibly specified. The OR-decomposition of a goal represents alternative ways in which a system can achieve the goal. Specifying tolerances for quality requirements gives a system room to maneuver and adapt. The flexibility that comes with using relaxed requirements comes at the cost of compliance—*a price not worth paying*.

Whittle et al. formulate an interesting relaxed requirement $\mathbf{W_3}$: *...EVENTUALLY, all devices SHALL use the same data* for a device synchronization application. I mention this requirement especially because it is formalized in traditional temporal logic unlike the ones I mentioned above which are formalized in a fuzzy temporal logic. The notion of eventual consistency sits well with the nature of distributed systems. At a first glance, therefore, it would seem that such requirements would be unavoidable for distributed systems. However, $\mathbf{W_3}$ is not crisp because *eventually* can be at any point in the future, and is therefore unbounded in time. No system will ever be noncompliant with $\mathbf{W_3}$. No temporal requirement of the nature *AFp* (on all paths, eventually p) is crisp.

III. CONTEXT

Instead of talking about critical versus noncritical requirements, it is far more beneficial to talk about *contextual requirements* in the sense of Ali et al. [7]. The motivating idea is that for any reasonably complex system, requirements can seldom be effectively specified without specifying the context in which a requirement would apply. Further, as the context changes, the system adapts to meet the applicable requirements. In their work, the context is understood as a property of the environment. Ali et al.'s produce Tropos-like goal models, except that the models are also explicitly annotated with the context. Another kind of model in their approach is a detailed model of the context itself. They start with abstract contextual conditions and refine them to events that can be monitored from the environment. Ali et al. [8] and Dalpiaz et al. [9] elaborate on adaptation driven by changes in context.

What Whittle et al. deem an invariant requirement may be better understood as a crisp requirement that is applicable in *all* contexts. A relaxed requirement may be better formulated as a crisp requirement that is applicable in *some specified* contexts. For example, in the smart home example, I could express the requirement that in the course of *normal* operation (a suitably modeled contextual condition), both $\mathbf{W_0}$ and $\mathbf{W_2}$ apply. That may be considered the default case. In the case of *abnormal* operation though (again, a suitably modeled contextual condition), $\mathbf{W_0}$ does not apply. If the abnormal condition is a power failure, that means that the system would adapt by using the backup generator towards the satisfaction of $\mathbf{W_2}$. How the system adapts may be left to the system (as a lower-level plan); however the specification of the abnormal conditions is part of the requirements specification.

Whittle et al. also consider, as part of the methodology of going from the invariant to the relaxed requirement, the contextual conditions that lead to the relaxation of the requirement. They also produce a mapping to monitorable events. However, differently from Ali et al., the exact conditions under which a requirement may be relaxed (the abnormal conditions) are themselves not part of the specification. Whittle et al. seem to want to avoid making the conditions part of the requirements specification in order that the system would have the freedom to adapt as suits the satisfaction of the invariant requirements. Doing that amounts to underspecification: there is no recourse to explicitly modeling the context as part of requirements specification.

IV. Uncertainty

In the literature, I often come across phrases such as *adaptation to unanticipated situations* (equivalently, unforeseen situations). Enabling such adaptation is an ambitious goal although, in general, I do not think we can engineer systems to handle unanticipated situations. What can we do is thoroughly analyze the environment and build our specifications in accordance, much as advocated by Zave and Jackson [10]. We can build contextualized representations of requirements so that the system can adapt to changes in context. We can build flexibility into our specifications. We can extensively test, verify, and simulate the system to make sure that the system works for the assumptions we have made. Part of coming up with a specification is thinking of all possible scenarios in which we need to the system to work.

Consider the recent partial meltdown at the Fukushima reactor in Japan. Modern nuclear plants are designed with elaborate safety systems to meet the requirement of preventing a reactor core meltdown. Designers envisage various threat scenarios and perform extensive simulations to make sure the safety systems meet requirements. However, the Japanese did not take into account a tsunami and an earthquake, both more powerful than their systems were built for, striking at the same time. This is what happened in Fukushima leading to a catastrophic core meltdown. Even though the system tried to adapt to cope with the circumstances (for example, by stopping the operation of the reactor and using alternate mechanisms to cool down the reactor), it simply wasn't built to handle the twin threat.

The lesson is that practical systems have limits: no matter how deep the analysis, engineers cannot design a system to cope with unforeseen circumstances. Your system is as only as robust as the assumptions you have made.

A conceptually helpful way to think of uncertainty is in terms of flexibility and compliance. Flexibility and compliance are two sides of the same coin. Higher-level abstractions enable more operational flexibility at the lower-level. However, flexibility cannot be unbounded, as is the case with relaxed requirements. There must also be a way to determine compliance. To satisfy $\mathbf{W_2}$ in case of a power failure, the system can either use a backup generator or immediately alert a human operator. As long as $\mathbf{W_2}$ is satisfied, it does not matter which. (The connection between compliance, flexibility, and abstraction in due to Yolum and Singh's work on business protocol specification [11].) Cheng et al. [1] bring up the issue of managing uncertainty. The idea is to prevent adaptation from producing *undesirable unforeseen* results. This possibility is much more distinct with relaxed requirements than with crisp ones.

V. Systems

I am interested in the class of information systems that support social and business interactions among multiple *autonomous*—broadly, independently motivated—parties. In settings as diverse as social networking, collaborative scientific computing, emergency response systems, cross-organizational supply networks, electronic commerce, and health care, to name but a few, we see that the systems are constituted from a number and variety of autonomous parties. I refer to each autonomous party as an *agent* and to the system as a *multiagent system*.

What I mean by autonomy above is different from what is understood as *autonomous operation* in the self-* approaches. I conceptualize an agent as a social entity that can be traced back to a real-world principal whose rationale it represents. The principal would be a human or an organization. Agents are autonomous because their principals are autonomous. By contrast, autonomous operation means the degree to which a software system can function without supervision. My vacuum cleaner may operate autonomously but it is not an agent in the above sense—ultimately, I control it. By contrast, bidders and sellers on eBay are all autonomous parties, that is, agents.

Because the agents are autonomous, they interact on the basis of social convention (just as humans do). A protocol is a specification of convention. Instead of referring to specific agents, a protocol is specified in terms of the roles that agent may adopt. Traditional computer science approaches for protocol specification are operational: they specify a flow of messages. They overlook the semantic content of messages. For example, traditionally a purchase protocol would specify that the accept or reject of an offer follow the offer; however, it ignores the fact that in many business settings an offer would, by force of convention, mean a social commitment from the seller to the buyer for goods in return for payment. Commitments are in fact a higher level abstraction for interaction: they abstract over message flow, similar to how goals abstract over low-level procedures [11]. Commitments also support compliance-checking: an agent is compliant as long as he does not violate his commitments to others.

Multiagent systems are logically distributed. Each agent in a multiagent system is a locus on control. There is no centralized computer that controls the actions of agents. You cannot integrate two agents; you must model the interactions among them. Each agent is independently designed starting from an independent set of requirements (an implication of autonomy) and therefore acts, interacts, and *adapts* independently of other agents in the system. The commitments that come about from enacting the protocol form the social fabric that connects agents. In adapting an agent would also consider his commitments with others. For example, if the winning bidder reasons that it will not be able to fulfill its commitment to pay on time, it may adapt by delegating payment to another agent or by offering an incentive to the seller in return for a delay in payment. The notion of compliance is relevant to adaptations here. Each agent would run his own monitor-deliberate-adapt loop independently of others. Normally, agents would not be reckless in their adaptations for fear of violating their commitments. Noncompliance will usually have bad social repercussions for an agent. Prudent agents would not arbitrarily make or cancel commitments when adapting; they would try to remain compliant. Dalpiaz et al. [12] propose a conceptualization of adaptation in multiagent systems that takes into account the commitments among agents.

When the software engineering community (including architecture and RE) discusses the engineering of systems such as flight control systems, washing machine, smart homes, vacuum cleaners, mission-critical safety systems, and so on, it takes a logically centralized perspective. RE approaches advocate eliciting requirements from different stakeholders; however, from the perspective of designing the system, there would simply be a unified pool of requirements that would have to be met by the system by performing the correct computations. This is what is traditionally referred to as *integration*. Approaches for architectural adaptation are likewise centralized: they assume a controller that manages the adaptations [3].

Logical centralization should not be confused with physical centralization. A logically centralized system does not mean that components in the system cannot be physically distributed. For example, a workflow engine is a logically centralized system that invokes distributed services; a flight control system would involve communication among different sensors.

Consider any system that involves two or more agents. I gave the example of auctions above, but it could just as well be scheduling a meeting, hosting a party, negotiation, argumentation, an interorganizational business process, or healthcare. To reason about adaptation in such a system, one would have no recourse but to model the interactions among agents. RE must adopt agents and commitments as first class ontological concepts. Recent work on core ontologies for RE [13] is crucially lacking in this regard.

VI. Conclusion

In this paper, I have dwelled on the nature of requirements specifications and systems as relevant to adaptation. I have argued that specifications should be crisp; in other words, one should be able to determine runtime compliance with the specification. I have also argued that rather than modeling requirements as critical or noncritical, one should model the context in which requirements are relevant. Further, I have argued that engineering for adapting to conditions unforeseen at runtime is a futile goal and that instead, one should focus on a careful analysis of the environment in coming up with specifications. And lastly, I have claimed that because current RE approaches take a centralized perspective on systems engineering, multiagent systems are outside the scope of what they can model.

I conclude by raising two further points.

One, in my vision, a requirements specification would ultimately be seen as a *program* that is interpreted by a virtual machine (VM). The VM computes a strategy for achieving the goals a specification denotes. It then puts the strategy in action, monitors the success of the strategy by observing events from the environment and tracking the achievement of goals, and if necessary computes and applies a new strategy. I dub the VM a *requirements engine* because what it is doing is essentially executing the requirements. Based on conversations I've had with several researchers, it seems to me that many consider the idea of requirements specifications as programs somewhat far-fetched. Their principal objection seems to be that too many details would have to be captured in the requirements models for the model to be executable, and that this would makes both the models and their analysis unwieldy. However, I do not see why the same objections would also not apply to any reflection-based approach for adaptation.

Two, problems with compliance aside, requirements such as $AF\mathit{messageDelivered}$ are more complex than they initially let on. Let's label this requirement R (R is a somewhat simplified form of the device synchronization requirement mentioned in Section II). Zave and Jackson [10] distinguish between requirements and specifications in the sense that a specification is implementable whereas a requirement may not be. $AF\mathit{messageDelivered}$ is a requirement, but it is not a specification. It is not a specification because message delivery is controlled by the environment. R would have to be further refined to arrive at a specification. One could make the domain assumption (K) that eventually some attempt to send a message will succeed. The specification (S) could say that messages be resent at periodic intervals until an acknowledgment is received. Informally, we can see that $S, K \vdash R$. If we want to support requirements-driven runtime adaptation, we should make sure what the system is given is a *crisp specification*.

Acknowledgments

The author gratefully acknowledges discussions with Munindar Singh and John Mylopoulos. Fabiano Dalpiaz, Raian Ali, Adina Sirbu, and Paolo Giorgini gave helpful comments on an earlier draft. The research was supported by a Marie Curie Trentino fellowship.

References

[1] B. H. Cheng, R. de Lemos, H. Geise, P. Inverardi, and J. Magee, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, ser. LNCS, vol. 5525. Springer, 2009, pp. 1–26.

[2] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for RE for self-adaptive systems," in *Proceedings of the 18th IEEE International Requirements Engineering Conference*, 2010, pp. 95–103.

[3] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[4] W. Heaven, D. Sykes, J. Magee, and J. Kramer, "A case study in goal-driven architectural adaptation," in *Software Engineering for Self-Adaptive Systems*, ser. LNCS, vol. 5525. Springer, 2009, pp. 109–127.

[5] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "RELAX: A language to address uncertainty in self-adaptive systems requirement," *Requirements Engineering*, vol. 15, no. 2, pp. 177–196, 2010.

[6] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*. IEEE Computer Society, 2010, pp. 125–134.

[7] R. Ali, F. Dalpiaz, and P. Giorgini, "A goal-based framework for contextual requirements modeling and analysis," *Requirements Engineering*, vol. 15, pp. 439–458, 2010.

[8] ——, "A goal modeling framework for self-contextualizable software," in *Enterprise, Business-Process and Information Systems Modeling*, ser. Lecture Notes in Business Information Processing, vol. 29. Springer, 2009, pp. 326–338.

[9] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "An architecture for requirements-driven self-reconfiguration," in *Proceedings of the 21th International Conference on Advanced Information Systems Engineering*, ser. LNCS. Springer, 2009, pp. 246–260.

[10] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 1, pp. 1–30, 1997.

[11] P. Yolum and M. P. Singh, "Flexible protocol specification and execution: Applying event calculus planning using commitments," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems*. ACM Press, 2002, pp. 527–534.

[12] F. Dalpiaz, A. K. Chopra, P. Giorgini, and J. Mylopoulos, "Adaptation in open systems: Giving interaction its rightful place," in *Proceedings of the 29th International Conference on Conceptual Modeling*, ser. LNCS, vol. 6412. Springer, 2010, pp. 31–45.

[13] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the core ontology and problem in requirements engineering," in *Proceedings of the 16th IEEE International Requirements Engineering Conference*, 2008, pp. 71–80.