Awareness and Adaptivity Requirements

John Mylopoulos
University of Trento

SEAMS’10, Cape Town
May 3-4, 2010
Abstract

Adaptive systems usually achieve adaptivity through a feedback loop, an architectural prosthesis that introduces monitoring, diagnosis and reconciliation functions to the system proper. We are interested in studying the requirements that lead to such feedback loop functionality. We introduce a class of requirements, called awareness requirements, which are best operationalized through a feedback loop instead of a collection of functions. These are characterized by the fact that they refer to other requirements, quality constraints or domain assumptions, and their success or failure. We then discuss adaptivity requirements as a special class of awareness requirements which talk about changes to the status of other requirements.
Thanks

This is on-going joint work with Alexei Lapouchnian (UToronto) and Vitor Souza (UniTN).

Thanks to my colleagues at the University of Trento for helpful comments and feedback: Raian Ali, Amit Chopra, Fabiano Dalpiaz, Paolo Giorgini, Nauman Qureshi, Angelo Susi.
Adaptive software systems

- Change their behaviour at run-time in response to changes in their environment.
- Adaptation mechanism consists of a feedback loop whereby the outputs of the system are monitored and compensation action is taken, if system is not fulfilling its purpose.
- ‘purpose’ = requirements,
  ‘not fulfilling its purpose’ ⇒ system is aware of success/failure of its requirements.
Baseline

FLEA+KAOS [Feather98], ReqMon [Robinson06], ...

Schedule meetings ➔ Dynamic analysis ➔ Vanilla RE ➔ Feedback ➔ Plan ➔ Feedback ➔ Plan

Plan
Our task

Schedule meetings

Schedule meetings + ??

Plan

Feedback

Plan
... in words

- Requirements as run-time objects to reflect on [Finkelstein08].
- To do so, we need to talk about the success/failure of requirements at run-time.
- Many proposals – [Letier04], [Whittle09], [Baresi10], ...
- Our perspective: (i) Awareness gives rise to the need for feedback; (ii) Model awareness requirements; (iii) Propose a new operationalization for requirements, specifically tailored to awareness.
Awareness

Awareness: Consciousness, sentience, ability to sense and respond to the environment.

Many types of awareness play a role in the design of software systems (security/process/context/location ... )

Important topic, not only in Computer Science.

In Philosophy, awareness plays an important role in several theories of consciousness. Our notion of awareness requirements draws on the distinction between higher-order awareness (awareness of one’s own mental states) and first-order awareness (awareness of things external) [Rosenthal05].
Requirements

Collect timetables
- By person
- By system

Collect

Get available rooms
- OR cp1
- OR cp2

By system

Find rooms
- OR cp3

Choose schedule
- OR

Schedule meeting
- AND

Rooms available

Pre/Post-conditions

Choice points

Domain assumption

Tasks

Schedule

Pre/Post-conditions

Good quality schedule

Softgoal

Goal

>70% participation

Quality constraint
Requirements + control parameters

Choose schedule

By person
By system

Collect timetables

Collect available rooms

Get available rooms

Find rooms

Schedule meeting

Good quality schedule

>70% participation

AND

OR

OR
Requirements and specifications

- Requirements include (hard) goals, softgoals, tasks and quality constraints with associated relationships.
- All are to be understood as generic (classes) to be instantiated for any specific requirements fulfillment task.
- A specification consists of a collection of tasks (T), domain assumptions (DA), quality constraints (QC), and control parameter values, which together fulfill root-level goals.
Specification space

- From a requirements model $M$ with $n$ OR-refinements, we can derive $O(2^n)$ specifications. Each alternative consists of making a choice for each OR-refinement. Control parameters add to this, possibly an infinite space.

- One of these specifications is implemented by the running version of a system. We’ll refer to this specification as $CurrentSpec$. 
Requirements at run-time

Collect timetables

Get available rooms

Choose schedule

Schedule meeting

AND

AND

AND

Collect available rooms

OR

OR

By system

Available rooms

(Parts of) CurrentSpec

Collect

Pre/Post-conditions

instanceOf

Good quality schedule

CurrentSpec

SEAMS'10 -- 13
Object lifetimes: Goals

Analogous FSMs describe the lifetimes of other requirement objects (QC, DA, T)
Uncertainty

Operationalizations here (and in all GORE frameworks) basically mean: If tasks T are carried out and quality constraints QC and domain assumptions DA hold, then requirements R are fulfilled.

\[ T, DA, QC \models R \]

That’s an optimistic view of the world …

- Rooms may not be available for meetings,
- Someone may not turn in his timetable, …

Executions of an operationalization may fail

\[ \Rightarrow \text{uncertainty!} \]
Adaptivity is all about governing/controlling uncertainty
Awareness requirements

- Refer to other requirements (Goals/Tasks/Quality Constraints/Domain Assumptions) and their success/failure.
- Consider

  r = ‘schedule meeting’, da = ‘always rooms available’

  r1 = ‘r will be completed within 2hrs’ (delta)

  r2 = ‘r won’t fail >3 times per year’ (aggregate)

  r3 = ‘avg r time won’t increase between months’ (trend)

  r4 = ‘da won’t fail >3 times per year’
Expressing awareness requirements

We need a language that allows us to express constraints on instances of a class ... An OCL-like language will do:

\[
\begin{align*}
\text{r} &= \text{goal class SM, da} = \text{domainA class RA} \\
\text{r1} &= \text{goal class R1, attr r: SM | dur(self.r) \leq 2hrs} \\
\text{r2} &= \text{goal class R2, attr r: SM | size(select x SM.inst st failed?(x) \land \text{within}(x,1yr,now)) \leq 3} \\
\text{r3} &= \text{avg(dur(select x SM.inst st within(x,1mo,now)))} \\
&\quad \leq \text{avg(dur(select x SM.inst st within(x,1mo,now-1mo)))} \\
\text{r4} &= \text{size(select x RA.inst st failed(x) \land \text{within}(x,1yr,now) \leq 3}
\end{align*}
\]
Strata of awareness

Of course, one could have awareness requirements about other awareness requirements.

For example,

- $r = \text{‘schedule meeting’}, \ da = \text{‘always rooms available’}$
- $r_1 = \text{‘$r$ will be completed within 2hrs’}$
- $r_1' = \text{‘$r_1$ won’t fail >3 times per year’}$

To avoid paradoxes, we need to stratify our requirements space: base requirements are at stratum 0, awareness requirements about base requirements are at stratum 1, ...

In general, an awareness requirement at stratum $n+1$ talks about a requirement at stratum $n$, and can only talk about requirements at strata $\leq n$. 
Adaptivity Requirements

An *adaptivity requirement* is an awareness requirement that can also talk about changes of status for another requirement.

For example, da = ‘always rooms available’
\[ r = ‘schedule meeting’, r1 = ‘r will be completed within 2hrs’ \]
\[ r2 = ‘r will be completed within 3hrs’ \]

Possible adaptivity requirements

\[ \text{dur}(r1.r.\text{collectTT}) > 2\text{hrs} \Rightarrow \text{fail}(r1) \land \text{initiate}(r2) \]  
\text{(relax)}

\[ 2.2\text{hrs} > \text{dur}(r1.r) > 2\text{hrs} \Rightarrow \text{fulfill}(r1) \]  
\text{(good-enough)}

\[ \text{dur}(r1.r.\text{collectTT}) > 3\text{hrs} \Rightarrow \text{fail}(r1) \land \text{fail}(r) \]  
\text{(abort)}

\[ \text{failed?}(r.da) \Rightarrow \text{initiate}(gmr) \]  
\text{(compensation)}

\[ \text{dur}(r1.r.\text{collectTT}) > 2\text{hrs} \Rightarrow \text{changePar}(FhM:=FhM-20\%) \]  
\text{(compensation)}
Adaptivity Reqs: Changing CurrentSpec

So far, examples of adaptivity requirements affect the status of requirement instances. But sometimes, we want adaptation to amount to a “from now on” change.

Suppose CurrentSpec assumes there are rooms available. If this DA fails more than 3 times within a month, we want to change choice point 2 for CurrentSpec and increase the # of rooms available for meetings (RfM):

\[
\text{size}(\text{select } x \text{ RA.inst st failed?}(x) \land \text{within}(x,1yr,\text{now}) > 3 \Rightarrow \text{changespec}(\text{cp2} \neq 1, \text{RfM}:=\text{RfM}+10)
\]

It would be useful to have variations of changespec that (i) maximize familiarity in the new spec relative to the old one, or (ii) minimize new implementation needed for the adaptation, etc. (many variations)
Reactive Operationalizations

Sometimes, a goal is fulfilled by simply waiting for its fulfillment. More generally, for a goal $G$ consisting of $G_1 \land G_2$, we may choose to do nothing until $G_1$ or $G_2$ are true (... opportunistic operationalization).

Let’s postulate that each awareness requirement (on another requirement $r$) is to be operationalized reactively with a feedback loop consisting of monitor, diagnose, reconcile, compensate tasks.

Diagram:

- M: Monitor
- D: Diagnose
- R: Reconcile
- C: Compensate

Feedback loop:

- $r$
Monitor-Diagnose-Reconcile-Compensate

- **Monitor** – the environment; usually applies to tasks, quality constraints and domain assumptions, *not* to goals.
- **Diagnose** – interpret monitored data; figure out the status of goals given information on the execution of tasks; see [Wang09], [Souza09] for principled ways of interpreting log data about tasks to offer a diagnosis about the failure of goals.
- **Reconcile** – what’s the adaptation, at the instance/class level.
- **Compensate** – change management, may include new implementation and/or adoption of results of aborted execution.
Operationalizations revisited

- We now have three types of operationalizations, as shown below.
- The first two are proactive, the last reactive.

![Diagram showing three types of operationalizations: Monitor-Diagnose-Compensate-Execute, with a blue circle, a yellow brain, and a blue hexagon.]
What operationalization, when?

- It might seem that proactive operationalizations are only appropriate for stratum 0 requirements, and reactive ones only for higher strata.

- Not so: either operationalization is in principle applicable to any requirement ...
  
  ✓ ... I schedule a meeting by waiting for someone to do it.
  
  ✓ ... I ensure room availability by cancelling meetings.

- The use of proactive operationalizations for stratum 0 requirements, and the reactive ones for higher strata is just a convention that design theories of the future may do away with.
Where do awareness reqs come from?

The need for adaptivity comes from criticality and risk considerations. Criticality, in turn, can have its origins in safety, dependability, reliability, etc.

Such non-functional requirements constitute the origins of awareness requirements.

Consider: Meeting scheduling (MS) is a critical requirement for our organization; hence we allocate more resources to MS, we do more V&V for our MS system, AND we impose some awareness requirements for it as well …
Stability

Our examples so far are simple in that they each set a bound on failures. Consider now awareness requirements such as

\[ r_1 = 'r \text{ will fail } \leq 10 \text{ and } \geq 5 \text{ times per month}' \]

Such requirements make sense in cases of tradeoffs: we want to control failure rates, but knowing that such control is costly, we also want a failure lower bound ...

Such requirements raise the spectre of oscillating adaptations, e.g., switching between manual to automatic collection of timetables, that violate \( r_1 \) wrt its lower/upper bound.
So what?

Schedule meetings

Schedule meetings

Awareness requirements

Feedback

Plan

Plan

Plan
Discussion

The idea of having requirements that refer to other requirements was there in the NFR framework [Chung00]; but the language used was very simple and there was no notion of reactive operationalization.

[Robinson08] uses an extension of OCL, OCL\textsuperscript{TM}, to talk about run-time requirements objects. This is probably the language we want to use for awareness requirements.

Recent work by Baresi et al [Baresi10] proposes adaptive goals that “... represent adaptation strategies to be performed when a goal is not fulfilled satisfactorily.” This seems to fit somewhere between our adaptivity requirements and reconciliation strategies in our proposal.
Conclusions

We have sketched a theory of software adaptivity founded on the concept of awareness.

The theory is based on a careful account of what awareness means in the context of requirements, how to express it, and how to operationalize it in terms of reactive plans.

The two major extensions required for GORE frameworks to account for that theory are: (i) Ability to refer to requirements elements within an OCL-like language; (ii) A new reactive operationalization primitive.

Given the complexity of the task-at-hand, we have strived to live by Einstein’s words: Make everything as simple as possible, but not simpler ...
Future work

- Three top issues to be addressed: semantics, semantics and semantics ...
- Stability
- Implementation
- Case studies ...
References


References (cont’d)