Self-explanation in Adaptive Systems

Nelly Bencomo
INRIA Paris-Rocquencourt
Le Chesnay, France
nelly@acm.org

Kris Welsh
School of Comp.
University of Kent, UK
k.welsh@kent.ac.uk

Pete Sawyer
Lancaster University, UK
INRIA, France
sawyer@comp.lancs.ac.uk

Jon Whittle
School of Comp. & Comm.
Lancaster University, UK
whittle@comp.lancs.ac.uk

***Abstract*—The behaviour of self adaptive systems can be emergent. The difficulty in predicting the system's behaviour means that there is scope for the system to surprise its customers and its developers. Because its behaviour is emergent, a self-adaptive system needs to garner confidence in its customers and it needs to resolve any surprise on the part of the developer during testing and mainteinance. We believe that these two functions can only be achieved if a self-adaptive system is also capable of self-explanation. We argue a self-adaptive system's behaviour needs to be explained in terms of satisfaction of its requirements. Since self-adaptive system requirements may themselves be emergent, a means needs to be found to explain the current behaviour of the system and the reasons that brought that behaviour about. We propose the use of goal-based models during runtime to offer self-explanation of how a system is meeting its requirements, and why the means of meeting these were chosen. We discuss the results of early experiments in self-explanation, and set out future work.**

***Index Terms*—self-explanation, self-adaptive, goals, claims**

I. Introduction

Self-adaptive systems posess an ability to adjust their behaviour in response to changes in their operating environment. Uncertainty in the nature of the operating environment may cause the behaviour of self-adaptive systems to be emergent. A system whose behaviour cannot be accurately predicted poses serious problems in terms of assurance and acceptance. Lack of intelligibility may cause users to stop using a self-adaptive system [1], [2], [3]. Because its behaviour is emergent, a self-adaptive system needs to garner confidence in its stakeholders, and allow developers to understand observed behaviour [3]. We believe that these two functions can only be achieved if a self-adaptive system is also capable of self-explanation.

We argue that a self-adaptive system's behaviour is best explained in terms of the statisfaction of its requirements. Observing the degree to which a system satisfies its requirements is well-discussed in requirements monitoring literature [4], and addresses questions of *what* the system is doing. The ability of self-adaptive systems to select alternative configurations based on environmental triggers raises questions on *how* the system is doing it, with more useful explanations offering clues to *why* the system is behaving as observed.

Readily-understandable explanations are challenging to produce, with several key challenges preventing developers from readily creating such functionality. These are discussed in the following paragraphs.

Firstly, an ability to explain behaviour relies upon an ability to monitor, introspect and reason about the system's current and past behaviour. There has been significant research interest in providing support for requirements monitoring [5], [4]. In the specific area of self-adaptive systems, advances have also been made towards better support for introspection by adaptive middleware [6], [7] and other frameworks [8], [9]. However, work seeking to combine these two capabilities with reasoning is in its infancy. The new and broader research area of requirements-aware systems covers similar interests [3], [10], [11].

Secondly, explanations need to be created at a sufficiently high level as to be understandable by a variety of interested stakeholders (e.g. end-users, but also by non-developers and support personnel). Ideally, users should interact with the system at a level of abstraction that is meaningful to them. This requires that the system is able to trace backwards and forwards between abstractions at the users level and abstractions used by the systems at lower levels (e.g. components, component configurations, etc.). A trace of relevant events in the history of the adaptations the system has gone trough should be kept by the system.

Thirdly, for self-explanations to be trustable, a self-adaptive system should be able to trace down from goals towards code to keep a synchronizationed link between requirements and architecture during execution. This trace needs to consider the dynamic changes that will affect requirements and the architecture of the system at runtime and keep a causal connection between the two.

Finally, a self-adaptive system should be able to reproduce a trace history of the adaptations it has performed in a way that is meaningful to support self-explanation.

In [12], we described our view of requirements-aware systems. In our work, representations of assumptions are made explicit using the concept of claims in goal models at design time. Using what we call claim refinement models (CRMs), we have defined the semantics for claims in terms of their impact on alternative strategies that can be used to pursue the goal of the system. The impact is calculated in terms of satisfaction and trade-off of the system's non-functional requirements (modeled as softgoals). Crucially, during runtime when the executing system monitors that a given claim does not hold anymore, the system may adapt to an alternative goal realization strategy that may be more suitable for the

new contextual conditions. Importantly, our approach tackles uncertainty, i.e. the new goal realization strategy may imply a new configuration of components that was not necessarily foreseen during design time. With the potential for non-foreseen behavior, self-explanation capabilities are crucial. In this paper we build on the approach described in [12] to address the challenges posed by self-explanation described above.

The rest of the paper is organized as follows: In Section II we present the motivation of the paper using a simple but yet useful discussion. In Section III, we discuss our initial progress towards a mechanism by which self-explanation can be achieved. In Section IV, we apply this means of providing self-eplanation to a short case study. Section V describes relevant related work. Section VI concludes the paper and discusses future work.

II. Motivating Example

Consider the example of a robotic vacuum cleaner for domestic apartments, which uses self-adaptation to balance two conflicting non-functional requirements: to avoid causing a danger to people within the apartment (*avoid tripping hazard*) and to be economical to run (*minimise energy costs*). The cleaner supports two modes of operation: clean at night and clean when empty. Cleaning at night will likely yield lower energy costs, but could cause the occupants to trip should they awake and move about the apartment. Cleaning when the apartment is empty eliminates this hazard, but if the apartment is only empty during daytime this will come at a cost of increased energy costs. A standard goal model, showing the different ways in which the robot can clean the apartment, and each method's impact on the two competing NFRs (which can be modelled as softgoals) would be deadlocked, with no clear favourable goal operationalisation strategy. We have previously discussed [13] the use of claims, which were first proposed in the Non-functional Requirements (NFR) Framework [14], to model an assumption made to break the deadlock in a goal model. In this case, we can make an assumption that the tripping hazard is unlikely to cause an accident. We illustrate this using an i* [15] Strategic Rationale (SR) model, which models how an agent achieves its goals, and allows alternative goal satisfaction strategies to be compared in terms of their impact on softgoals. The model in Fig. 1 shows a claim "No Tripping Hazard" breaking the deadlock that would otherwise occur.

In Fig. 1, the vacuum cleaner's "Clean Apartment" goal may be satisfied either by the "Clean at night" task, or the "Clean when empty" task. Cleaning at night *helps* satisfy the "Minimise energy costs" softgoal, but *hurts* the "Avoid tripping hazard" softgoal, as represented by the *contribution links* attached to the task. The "Clean when empty" task makes the inverse contributions to each of the softgoals.

The "No tripping hazard" claim *breaks* the negative contribution made to the "Avoid tripping hazard" softgoal by the "Clean at night" task, which means that this contribution should be lent less credence, or disregarded completely when

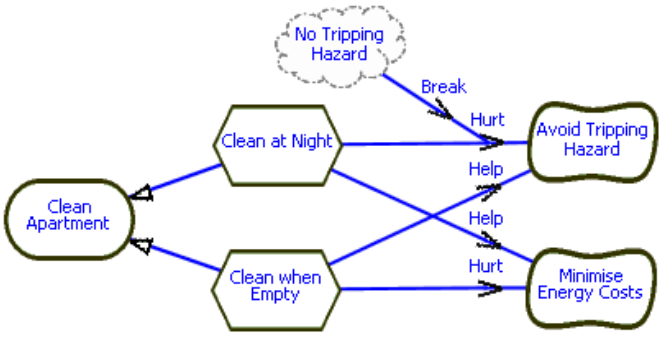

Figure 1. Goal Model of a Robot Vacuum Cleaner from [16]

deciding between the competing operationalisation strategies. With this assumption made, the decision to clean at night follows naturally.

Although assuming that the tripping hazard doesn't pose any real risk makes for a convenient way to break the deadlock in the goal model, the assumption is mere conjecture and would prove difficult to verify at design time. Thus, the robot vacuum cleaner is provided with a means of verifying the assumption at run-time, using monitoring. The broad nature of the "No tripping hazard" claim makes it more dfificult to identify a suitable monitoring mechanism, so we use a *claim refinement model* to decompose the claim hierarchichally into its underlying assumptions, until some more precise, and crucially monitorable, assumptions are identified. We consider a claim refinement model to be sufficiently complete when all leaf claims are either: monitorable, axiomatic or considered an unmitigatable risk. In the latter case, the claim marks the edge of the contextual envelope in which the system is capable of tailoring itself to suit.

In this example, our "No tripping hazard" claim has the CRM shown in Fig. 2. There are four sub-claims organized in two ANDed branches (claims may also be OR-ed). Together, the branches illustrate the rationale for why the root claim should hold. In this case, "No tripping hazard" holds if there is no-one in the room in which the vacuum cleaner is working AND no external impact is detected by the vacuum cleaner. The leaf claims of the CRM, "Light level [remains] constant" and "No shock detected" must be directly monitorable via events or statistical data collected by the system (they are monitorables). If a monitorable turn out to be false, for example, if the vacuum's inertial sensor detects an external shock, then claim falsification propagates upwards towards the root. In the case of the no tripping hazard CRM, the impact event would falsify the "No tripping hazard" claim. Similarly, a sudden increase in the light level would indicate that a light has been switched on by a waken occupant.

With a means of run time verification for the deadlock-breaking "No tripping hazard" assumption having being found, the robot vacuum cleaner can be specified as using a clean at night strategy unless a shock is detected or a light is switched on, in which case the robot should self-adapt to use the "Clean

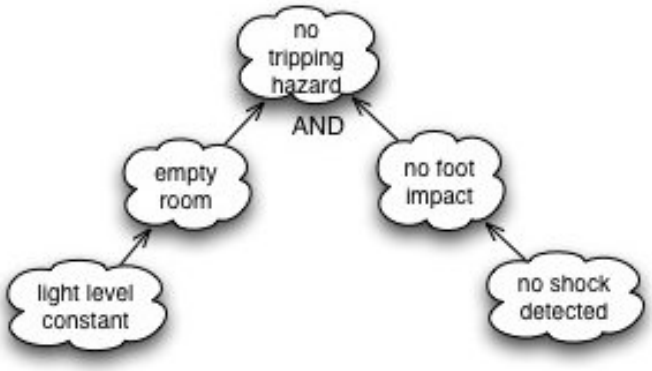

Figure 2. Claim Refinement Model for Robot Vacuum Cleaner

when empty" strategy.

However, after it has been in operation for some time, the owners of the robot vacuum cleaner find that it is costing more to run than expected. A self-explanation capability would mean that the vacuum cleaner could explain that it is required to avoid causing a tripping hazard, and that it has been unable to clean at night because the occupants frequently wake up and turn the lights on. In this scenario, the explanation would help the users to understand the system's behaviour, and help to pinpoint the reason the system is not behaving in the manner they would have imagined. The customer understands the reasoning, but is still dissatisfied because the operating costs are unacceptable. They submit a change request to the developer for the vacuum cleaner to be modified so that it only adopts the clean when empty strategy if two consecutive nights' cleaning have been interrupted.

In isolation, this change request may seem unimportant to the developers, especially if the change request is scant on background information justifying it. To contextualize the request, they interrogate the vacuum cleaner to determine its history of operation, with special attention to its history of self-adaptation and the events sensed in its environment that triggered adaptations. They discover the light detection event is being triggered more frequently than expected, and understand by consultation of the requirements model that this is interpreted as invalidation of the assumption that underpins prioritization of energy cost minimization.

The developers realize that running costs are high but note also that the customer does move around the apartment at night. They modify the vacuum cleaner's software to adopt a new strategy; they relax [17] the clean apartment goal by accepting that the clean apartment goal may be satisfied at a later time. The user change request is accepted; when interrupted, the robot tries to clean the following night before resorting to the clean when empty strategy.

In this simple example, the information contained within the explanation offered by the system could be obtained by analysis of standard debugging output or logs, and by deduction. However, these sources of information are low-level artefacts of particular code execution paths, and such analysis is performed by the system's developers, who will need time to perform the analysis. The potential for a self-adaptive system to adopt an unexpected configuration, or adopt an expected configuration in unexpected circumstances, means that there is a need for users to be able to understand what the system is doing, and why.

Our interest lies in reconciling a higher-level trace of the system's behaviour with its requirements, to establish whether the system's behaviour is appropriate, or better optimal, and whether the requirements themselves are correct. Although an explanation in terms of requirements may still prove too complex for some users to be able to understand a system's operation in some circumstances, the higher-level explanation may allow non-developer support personel to resolve queries without requiring developer input.

III. Self-Explanation Through Run-Time Requirements Models

Andersson *et al.* propose a means of characterising the change a self-adaptive system is designed to tolerate. Changes can be *foreseen*, *foreseeable* or *unforeseen* [18]. We ignore here systems dealing with unforeseen change, which are more properly a topic for artificial intelligence research and pose a different order of challenge both for self-adaptation and self-explanation.

Much of our previous work has concerned requirements modeling for systems dealing with *foreseen* change [19] [13] [16]. Where change is foreseen, the set of contexts that the system may encounter are known at design time. Here, a self-adaptive system can be defined as a set of pre-determined system configurations that define the system's behaviour in response to changes of environmental context. Thus, there is little or no uncertainty about the nature of the system's environment and, if it is developed to high quality standards, satisfaction of the systems requirements should be deterministic.

More recently [12], we have started to address systems dealing with change that is, in [18]'s terms, merely *foreseeable*. Here, they key challenge is uncertainty, where at design time some features of the problem domain are unknown, perhaps even unknow*able*. Crucially, and in contrast to unforeseeable change, the fact of this uncertainty can be recognized, offering the possibility of mitigating it by resolving the uncertainty at runtime. The uncertainty associated with foreseeable change typically forces the developers to make assumptions in order to define the means to achieve the system's requirements. Thus, for example, a particular environmental context may be assumed to have particular characteristics and the system's behaviour defined accordingly. If the context turns out to have different characteristics, the system may behave in a way that is inappropriate. This has led us to exploit the concept of markers of uncertainty. Markers of uncertainty serve as an explicit marker of an unknown that forces the developer to make an assumption. We implement markers of uncertainty using claims as described in the previous section. A benefit of using claims to represent design-time assumptions is that

the uncertainty is bounded and thus the risk of the system behaving in an inappropriate may be mitigated by monitoring, claim and goal evaluation, and adaptation.

Our solution uses i* goal and claim refinement models, as depicted in Figs. 1 & 2. As described in the previous section, claim monitoring may permit assumptions to be verified during operation. Where a claim turns out to be false, the corresponding portion of the goal model can be re-evaluated at run-time. If, as a consequence of this, the original goal operationalization no longer evaluates as the optimal solution, an alternative goal operationalization can be substituted dynamically, using the system's adaptation mechanism. We have applied our work to the domain of wireless sensor networks where our run-time models are supported by advanced adaptive middleware and domain-specific component models [6].

In the context this paper, the key feature of foreseable change is that it may result in behaviour that is emergent. Emergent behaviour may surprise stakeholders who may require the behaviour to be explained in order to build and maintain their confidence in the system. Our thesis is that the same run-time requirements models that we employ to handle unforeseen change can also be employed as the basis of a self-explanation capability. Partially based on [20], we characterize self-explanation for such systems as follows:

why = {what, how, history}

Where, **why** represents the explanation for **what** was observed, in which what was observed was a consequence of **how** the system satisfied its requirements, when interpreted using the **history** of adaptation events that have occurred. In the next section, we illustrate how the what, how, history and why of a system's behaviour may be provided using our run-time requirements models solution for the GridStix wireless sensor network.

IV. Case Study

To demonstrate self-explanation in the context of a system which adapts to contexts not fully foreseen, we present the GridStix flood prediction system [21]. We have previously discussed this system in the context of requirements modelling [13], and have recently been exploring run-time uses of these requirements models. In [12], we discuss systems using run-time goal-based models to guide adaptation to circumstances where assumptions on which the originally prescribed configuration(s) rely no longer hold. In this paper, we show how claims and run-time requirements models that have been implemented for GridStix support self-explanation.

The GridStix system is a wireless sensor network (WSN) for detecting and predicting flooding, versions of which were deployed on the river Ribble in North-West England and on the River Dee in North Wales. GridStix comprises a number of nodes (14 on the Ribble installation), each of which are equipped with sensors for detecting water depth and flow rate. The captured sensor data is processed by a stochastic model of the river to predict future river state. A feature of this algorithm is that it is distributed and lightweight enough to be executable by the GridStix nodes. Incremental results are cascaded from the most up-stream node down to the gateway node and from there via a GSM link to Lancaster University. Its accuracy is a function of the number of nodes contributing data.

GridStix nodes rely on batteries and solar panels for power, thus energy conservation is a key non-functional requirement. GridStix uses an ad-hoc overlay network in which nodes can communicate using Bluetooth or WiFi, configured as either a shortest-path or fewest-hop spanning tree.

To help test feasibility and derive requirements for GridStix, empirical data was collected from experiments with of a laboratory-based prototype. Data was collected to measure (among other metrics) resilience and power consumption [6], as illustrated by the graphs in Fig. 3. Here, resilience is a measure of network fragmentation; the more nodes become isolated from the gateway (uplink) node, the less resilient is the network. Power consumption measures per-hop power consumed during the transmission of 1KB of data from each node to the gateway. The graph *Physical Network Resilience* in Fig. 3 shows that the greater range of WiFi meant that data from each node could be routed to the gateway by a larger number of paths with WiFi than using Bluetooth, while the graph *Physical Network Power Consumption* in Fig. 3 shows that the additional resilience comes at the cost of higher power consumption.

Similarly, the graph *Spanning Tree Resilience* shows that, for a small number of nodes (nodes B, H and I), the number of routes to the gateway affected by node failure is much higher when using a shortest-path (SP) spanning tree algorithm than when using a fewest-hop (FH) spanning tree. The graph *Spanning Tree Power Consumption* shows that for the nodes furthest from the gateway node (nodes L, M, N and O) the power consumed in transmitting the data is significantly higher for a FH than SP spanning tree.

In other words, GridStix was predicted to be relatively resilient to node failure when configured to use WiFi and a fewest hop spanning tree, but at the cost of high power consumption.

Resilience and power consumption were two of GridStix's important non functional requirements. However, as shown in the experiments it is hard to optimize for both, meaning that one would have to be prioritized over the other. However, a feature of self-adaptive systems is that the extent to which any NFR must be satisficed tends to be context-dependent, and this was the case with GridStix. Goal-based models, and specifically soft goals, support reasoning about tradeoff decisions that are aimed at achieving optimal goal satisfaction.

For the purposes of GridStix, expert environmental scientists had partitioned river behaviour into three distinct operating conditions (*domains*); *quiescent*, *high flow* and *flood*. Quiescence was predicted to be the most common domain over time and so, with the need for the nodes to retain enough power to react when the river state changed, energy efficiency was the priority. When in the flood and high flow domains, by contrast, resilience was prioritized to better tolerate any node loss that

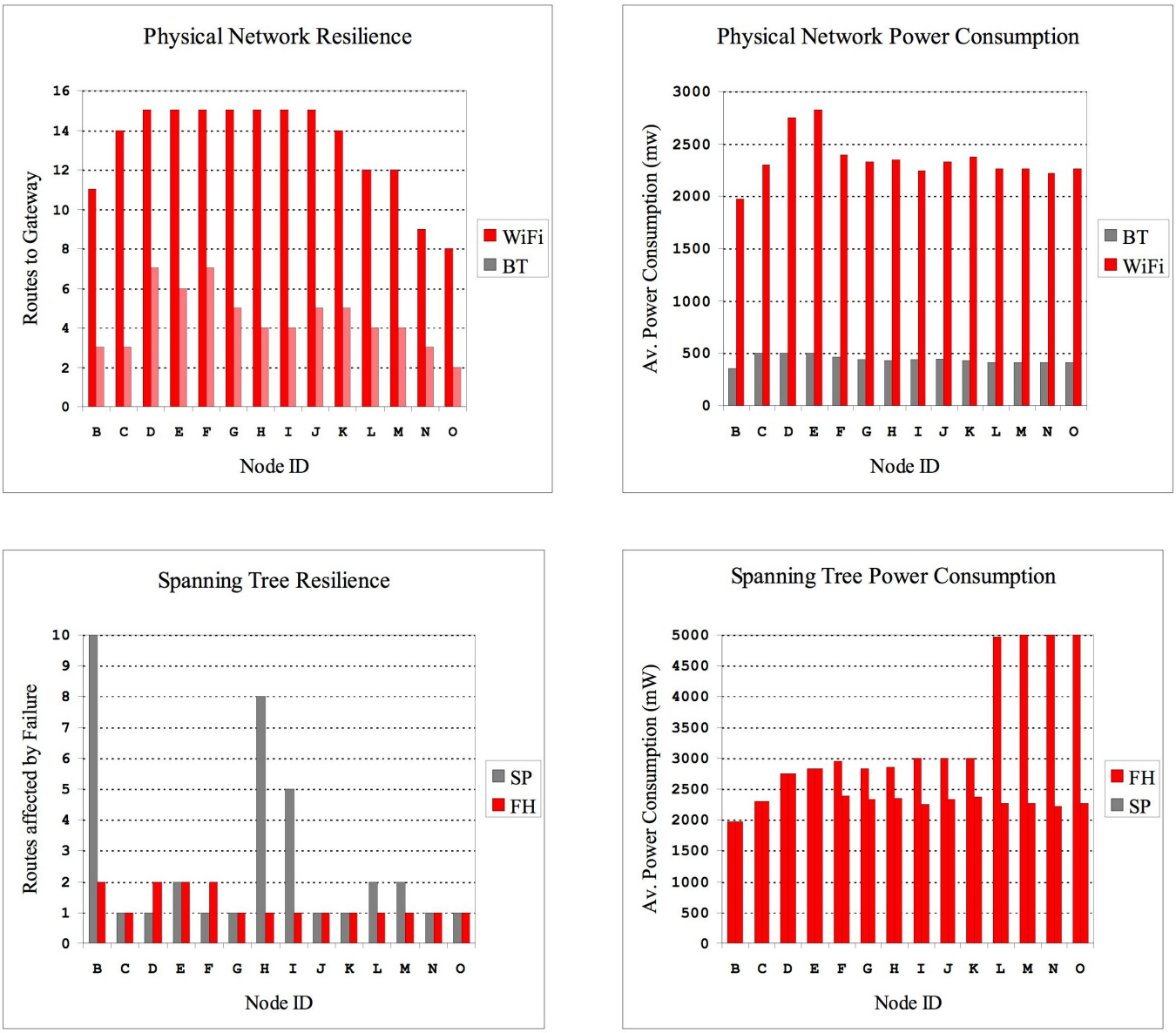

Figure 3. Laboratory Performance Data (reproduced from [6])

could impair the accuracy of GridStix's flood predictions. Thus, a particular GridStix configuration was specified for each domain, with (what was predicted to be) adaptation from one configuration to another specified to happen when the river was observed to change from one domain to another. These domain changes were based on sound knowledge and were therefore *foreseen*, meaning that we knew that the river's state would change and could specify the behaviour required of GridStix for each domain. Fig. 4 shows the goal model for the flooding domain (which we call S3). The figure shows the claims "Bluetooth too risky for S3", "SP too risky for S3" and "Single node image processing not accurate enough for S3". Each claim records an assumption about a design-time choice of goal operationalization, made because of uncertainty about the relative performance of alternative operationalizations in the field. The tasks (goal operationalizations) chosen are in white (i.e WiFi, and FH). Note that for simplicity reasons the single-node and multi-node image processing shown in the figure is not part of the explanation. However, similar conclusions can be made if we take into account these operationalizations and their effect on the NFRs, therefore the *calculate flow rate* goal should be ignored in the figure.

The configurations that were specified at design-time for each domain were based on the performance of the alternative communication technologies and spanning tree configurations observed in the laboratory experiments described above. However, we were aware that the lab results might prove imperfect predictors of how GridStix performed in the

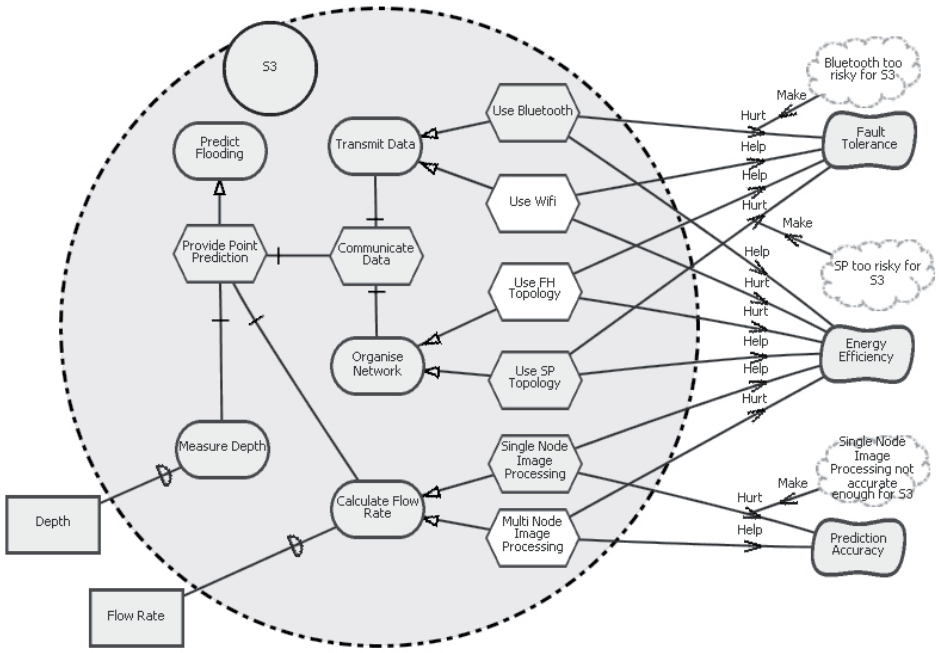

Figure 4. Gridstix goal models for the flooding state of the river

field. The initial River Ribble deployment confirmed that the effects of radio signal absorption by the river banks, rain, trees, etc., had a significant affect on performance [21]. To make GridStix more tolerant of these effects, it was augmented with claims to monitor the design-time assumptions, and to adapt to an alternative configuration if monitoring suggested that the alternative configuration could perform better. This was an important change because it meant that, in addition to the changes foreseen by knowledge of the different river domains, change as a consequence of operational experience was also *foreseeable*. When using claim monitoring, GridStix can decide by itself to adapt to a new configuration under some circumstances that were not predefined at design time. Thus, whereas GridStix's *adaptive* behaviour had been deterministic (even if its adequacy as a WSN had not been), its adaptive behaviour was now non-deterministic. Such non-deterministic behavior could cause "surprise" to an operator of the system, and therefore a self-exaplanation capability is appropriate.

A portion of the claim refinement models used by the GridStix *flood* and *high flow* domains is presented in Fig. 5. There is one top-level claim shown (in bold). This represents assumptions derived from the laboratory experiments that Bluetooth communication technology is too risky. In other words, the assumption is that if GridStix was configured to use Bluetooth, network resilience would likely be poor; implicitly poor*er* than if WiFi was used instead. The associated claim refinement model represents derivation of the means to sustain the claim and results in (using the labels in Fig. IV as shorthand for the subclaims):

$$\text{BT\_Too\_Risky} \Leftrightarrow (A_0 \Leftrightarrow (A_1 \vee A_2)) \wedge (B_0 \Leftrightarrow (B_1 \Leftrightarrow (B_2 \vee \neg B_3)))$$

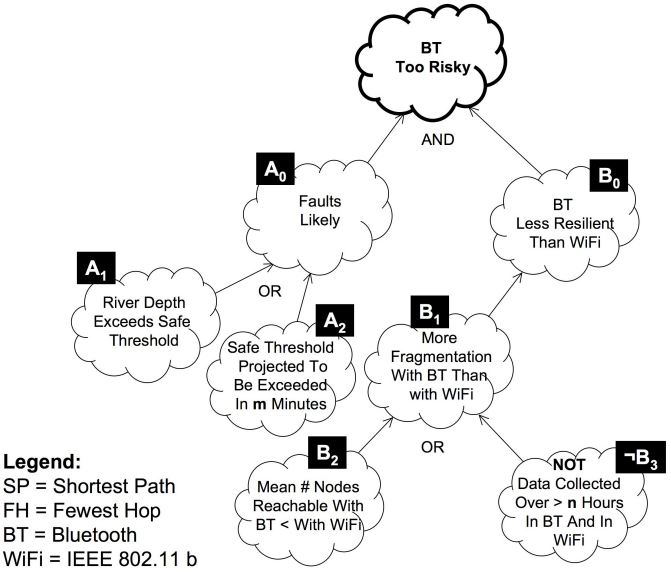

Figure 5. GridStix Claim Refinement Model Justifying Choice of WiFi for Inter-Node Communication

Thus, our root assumption, that using Bluetooth will lead to greater fragmentation than using WiFi (the *BT Too Risky* claim), will be disproved if any of the leaf (*monitorable*) subclaims is negated. In other words, Bluetooth is not likely to fragment the network if the river depth is below the safe threshold level or, at the current rate of change it will not exceed the safe level anytime soon. Similarly, Bluetooth is unlikely to lead to excessive fragment the network if the rate of fragmentation when using Bluetooth is no higher than when

using WiFi or, if there is data that contradicts this, there is too little data to make the contradiction statistically sound.

Because the River Ribble deployment of GridStix has been decomissioned, we used a simulator to observe the system's behaviour when experimenting with claim monitoring. The simulator has been developed using the collected data of the several months GridStix was deployed with the advantage that we can run experiments when needed. The simulator handles factors such as: power usage by batteries of nodes and according to whether the nodes were configured to use WiFi or Bluetooth, fewest hops or shortest path; whether the nodes were idling or performing computationally intensive tasks; and power replenishment from solar panels depending on time of day, amount of sunlight received or how cloudy the weather is, among others. Using a simulator constructed for GridStix, we ran an experiment to compare the longevity of the claim-augmented version of GridStix with the original. Longevity in this context means the length of time during which a sufficient number of nodes were connected to allow a meaningful result to be returned by the gateway. The simulator includes randomization to simulate jitter and packet loss. We complemented this with random node failures to simulate those actually observed. We ran the simulator with a profile of river behaviour over a fixed period comprising a sequence of flow rate and depth values that simulated the river in every mood from quiescent to flood. We varied a single variable; the amount of sunlight received by the nodes' solar panels, using percentage of cloud cover during daylight hours as a proxy. The experiment was run three times and the results averaged to account for the randomization elements.

The experiments suggest no benefit from claim augmentation when cloud cover is above approximately 40%. Once cloud cover drops below 40%, however, the augmented version has significantly greater longevity. For example, at 30% cloud cover, instead of failing after approximately 180 hours of operation, GridStix survives for approximately 250 hours.

What was observable was a change in GridStix's longevity. The **history** of GridStix's runtime adaptations reveals a correlation between the improved longevity and **how** it had been achieved; the substitution of Bluetooth for WiFi communication when the river was in high flow or in flood. The **history** of the monitoring data shows that over the defined minimum period for accumulating data, network fragmentation was no less during that period when using Bluetooth than when using WiFi. This in turn provides the **why**; the effect of this on the claim refinement model (Fig. 6) in which the falsified monitorable claims

$B_2 \vee \neg B_3$ ... became ... $\neg B_2 \wedge B_3$

and propagated up the hierarchy to falsify the top-level *BT Too Risky* claim that justified the original (design-time) choice of WiFi over Bluetooth. This in turn triggered the run-time re-evaluation of the goal model, revealing that the operationalization of the *Transmit Data* goal now favoured the use of Bluetooth rather than WIFi because Bluetooth's

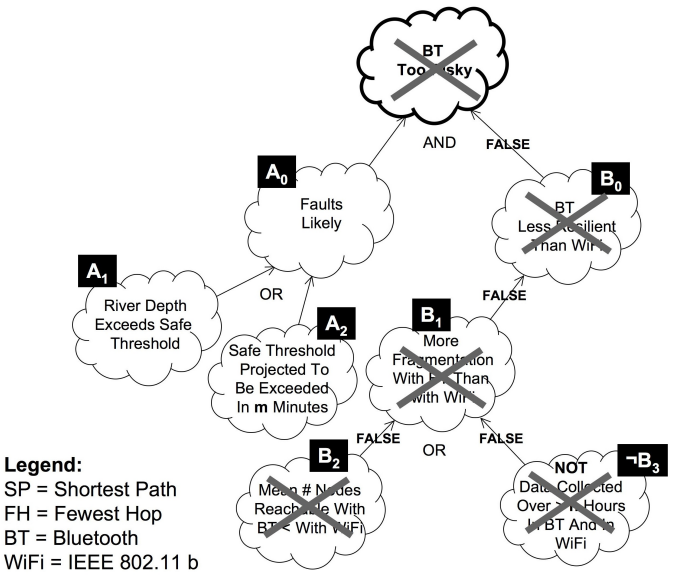

Figure 6. Falsified Claim Propagation

net impact on power consumption and resilience had become more +ve (positive) than that of WiFi. The goal model was thus changed to select Bluetooth as *Transmit Data*'s operationalization which in turn triggered the GridStix middleware to adopt a new component configuration, dynamically binding the Bluetooth component in place of the WiFi component.

Discussion

Revisiting the challenges presented in the introduction of this paper we conclude that our approach:

1. Offers suitable monitoring capabilities for self-explanation through the use of claim monitoring. As for reasoning capability, our claim refinement models allow a change of configuration to be traced back to the monitored, and falsified, assumption that caused it.

2. Can offer self-explanation and discourse at the level of requirements. Self-adaptation (that maybe misunderstood by operators) can be explained in terms of goals, operationalisations and assumptions. This level of abstraction is closer to natural language than architectural or code-level descriptions, offering more understandable explanations.

3. Provides the required link between the requirements and the architecture. The currently active configuration, at an architecture level, is linked to goal operationalisations, to the goals they achieve and their expected impact on system NFRs. (e.g. a proper link between the architectural use of BlueTooth and the "Communicate Data" goal it achieves, and the effect on system NFRs such as power consumption).

4. Finally, allows the system to be able to reproduce a trace history (i.e. a sequence of steps) of monitored events, assumption falsifications and resultant reconfigurations to explain the reasons behind a self-adaptation being carried out.

V. Related Work

Although, to our knowledge, we are the first to discuss self-explanation in the context of self-adaptive systems; the desire for systems to produce output at a higher level to improve understanding is long-standing. For example, there has been significant research into Natural Language Generation by the Artificial Intelligence and Computational Linguistics communities.

In [22], Duggan and Bent present an algorithm, designed to infer the type of variables during compilation of programs written in implicitly typed languages such as ML or Haskell, where explicit variable type declarations are not used. The algorithm infers variable type by analysis of variable usage, annotating the program's syntax tree as it progresses. Inference is performed using a set of rules; for example a variable to which the addition operator is applied, with a right hand operand of 1, is an integer. A variable whose type is determined to be integer through this example rule would have the following explanation annotated to the program's syntax tree: +(x,1) gives x: int. Explanations can become considerably more complex when a variable's type is dependent on that of one or more other variables, however the base format remains the same. In this work, the explanation is used by the algorithm itself to allow explanation fragments previously generated to guide later type inferences, but the authors consider the approach potentially useful in providing debugging support.

Similarly, in [23], Van Baalen *et. al.* retrofit a domain specific code generator with explanatory capability for use at NASA. In this work, the explanation covers the relationship between a specification, domain theory and synthesised code. The explanation is relatively low-level, designed to allow developers to prove correctness, given NASA's obvious need for high-assurance software.

In [24], Huang and Fiedler discuss the PROVERB text planner, which verbalises mathematical (natural deduction) proofs. The planner uses a three-stage approach, with the first stage responsible for hierarchichally decomposing complex proofs into a series of subproofs, the second stage identifies possible opportunities to "paraphrase" (or rather combine proof elements into larger, useful sentences) with the third stage actually generating the textual output. A more general overview of the state of the art in automated theorem provers, including discussion of the usefulness of their output, is offered in [25].

Although this work shows a research interest in providing high-level output to ease human understanding, our focus is not on programs providing natural language output, but in providing explantions of observed behaviour. Furthermore, the explanations offered by [22] and [23] are aimed at developers and mathematicians, respectively. The self-explanations we advocate are at a higher-level of abstraction, aimed at users and support personell.

Debugging mechanisms, even those considered high-level [26] [27], are focussed on data structures and code rather than on requirements, goals and operationalisations. More closely-related work can be found in the field of requirements monitoring [5] [4], from which we derive our claim monitoring. [28] proposes "awareness requirements", which are requirements that refer to the success or failure of other requirements. The authors state that awareness requirements may refer to goals, tasks, quality constraints and domain assumptions. Claim monitoring in our work is similar to domain assumption awareness requirements in [28], but their focus is on the mapping from requirements models to feedback loops, with no run-time representation of the awareness requirements.

The claim reasoning we use to demonstrate the utility of run-time requirements models in offering self-explanation is based on a combination of two previous streams of our work. In [13], we discuss the use of claims to highlight assumptions made during self-adaptive system specification, with a view to them being revisitted in light of later requirement changes. In [10], we make the case for the run-time use of requirements models, with the ability to rectify deficiencies in requirements satisfaction using self-adaptation being a key motivator. Although we use reasoning of run-time claim refinement models to offer a limited form of self-explanation, the type of self-adaptive system claim reasoning proves most useful for are those with a limited number of potential goal operisational strategies, or where self-adaptation is being used to balance a set of conflicting non-functional requirements.

Approaches such as RELAX [17] and FLAGS [29] adopt fuzziness in requirements to allow self-adaptation to prioritise and optimise their satisfaction. In these approaches, a run-time requirements model could be used to record the (re)prioritisations that take place, and to allow explanations of adaptations in the context of which requirements were compromised and which favoured. Approaches such as [30], which use KAOS [31] goal models, could benefit from run-time analysis of obstacle models to offer self-explanation in terms of which obstacles have been detected in the operating environment, and which goal operationalisations have been adopted to overcome them.

When tackling uncertainty, the ideas discussed in [32] are also related. As we do, the authors of [32] argue that uncertainty plays an important role in any software based system that needs adapt continuously to meet the goals. They argue that the focus of managing uncertain information should be on the rationale used to come to a decision. We emphasize the importance of being able to explain this rationale. In their case, the decision may be taken either during design or requirements (i.e. before execution). In our case, we go further because the self-adaptive system is able to make decisions at runtime as well. Finally, we believe our work is relevant to the implementation of dynamic traceability needed when dealing with self-adaptive systems where little work has yet been done. The authors of [33] discuss traceability in the presence of uncertainty. Similar to our work on claims, the authors of [33] propose to attach supplementary information to traceability links. This additional information describes the confidence and the rationale for its creation. The authors take into account the fact that the rationale that supports design decisions is often based on assumptions and beliefs. However,

in contrast to our work, their work focuses on the case of software product lines and their evolution during software life cycle rather than on runtime adaptation

VI. Conclusions and Future Work

This paper has argued that self-adaptive systems with the potential to behave in a manner not prescribed at design-time require self-explanation to allow emergent behaviour to be diagnosed, understood and explained. Self-explanation is important because it provides a means to increase confidence in, and resolve queries about, the behaviour of a self-adaptive system by its users. Self-explanation can also aid developers in understanding the behaviour of a self-adaptive system by tracing observed run-time behaviour (the *what*) to design-time assumptions, instrospect the strategy chosen (the *how*) and the extent to which they proved to be valid in operation (the *why*).

As already described in [12], [16] we have developed an approach to creating self-adaptive systems capable of tailoring their behaviour to an operating environment not fully foreseen at design-time, using run-time requirements models. These systems are capable, indeed likely, to exhibit emergent behaviour. In this paper we show how self-explanation of such behaviour might be generated from the systems' adaptive reasoning machinery. The particular run-time requirements models used by our approach are in-memory representations of i* Strategic Rationale and NFR framework Claim Refinement Models, which are notably high-level in their nature. Our hypothesis is that these dynamic models, interpreted through the history of observed behaviour and adaptation events can provide a plausible means of explaining why the observed behaviour came about. This contrasts with the use of low-level reconfigurations and executed code paths used in standard debugging tools which are difficult to interpret in terms of systems' requirements, even for expert developers working on systems that don't have the added complexity of a self-adaptive capability.

There are several ways in which our approach can be improved. Currently, the claim reasoning and model transformation based adaptation mechanism discussed in this paper applies where goals are achieved by selecting from a finite number of operationalization strategies defined a-priori but selected dynamically. Our approach is able to improve the flexibility of an executing system facing unforeseen situations, but the potential operationlization strategies, and the goals they achieve are defined and analysed at design time. Where new operationalization strategies may themselves be emergent (e.g. through dynamic service discovery), further research is needed. This is one of the topics we are investigating in the FP7 CONNECT project [1]. Specifically, we are studying ways in which a new operationalization strategy can be conceived at runtime. One of the challenges explored is that of updating goal models during execution to keep the required causal link between architecture and requirements.

[1] http://connect-forever.eu//

Acknowledgment

This research is partially supported by the Marie Curie Fellowship "Requirements@run.time" and the EU CONNECT project. Pete Sawyer has been partially funded by a visiting researcher grant to INRIA.

References

[1] B. M. Muir, "Trust in automation: Part i. theoretical issues in the study of trust and human intervention in automated systems," *Ergonomics*, vol. 37, no. 11, pp. 1905–1922, 1994.

[2] A. C. Gillieis and A. Hart, "Using kbs ideas in image processing - a case study in human computer interaction," in *Research and development in expert systems V: proceedings of Expert Systems '88*, 1988, pp. 258–268.

[3] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," *Requirements Engineering, IEEE International Conference on*, vol. 0, pp. 95–103, 2010.

[4] W. Robinson, "A requirements monitoring framework for enterprise systems," *Requirements Engineering*, vol. 11, no. 1, pp. 17 – 41, 2005.

[5] S. Fickas and M. Feather, "Requirements monitoring in dynamic environments," in *Second IEEE International Symposium on Requirements Engineering (RE'95)*, 1995.

[6] P. Grace, D. Hughes, B. Porter, G. Blair, G. Coulson, and F. Taiani, "Experiences with open overlays: A middleware approach to network heterogeneity," in *submitted to Eurosys 2008, Glasgow, UK*, 2008.

[7] G. Coulson, G. Blair, P. Grace, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, "A generic component model for building systems software," *ACM Transactions on Computer Systems*, February 2008.

[8] 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.

[9] J. C. Georgas, A. van der Hoek, and R. N. Taylor, "Using architectural models at runtime to manage and visualize the adaptation process," in *Models@run.time, Special Issue*, N. Bencomo, G. S. Blair, and R. France, Eds. IEEE Computer, 2009.

[10] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 199–202. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810329

[11] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness requirements for adaptive systems," in *ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24, 2011*, 2011, pp. 60–69.

[12] K. Welsh, P. Sawyer, and N. Bencomo, "Towards requirements aware systems: Run-time resolution of design-time assumptions," in *in proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*, 2011.

[13] K. Welsh and P. Sawyer, "Requirements tracing to support change in dynamically adaptive systems," in *REFSQ*, 2009.

[14] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Springer, 1999, vol. 5.

[15] E. S. K. Yu, "Towards modeling and reasoning support for early-phase requirements engineering," in *RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Washington, DC, USA, 1997.

[16] K. Welsh and P. Sawyer, "Understanding the scope of uncertainty in dynamically adaptive systems," in *REFSQ*, 2010.

[17] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel, "Relax: a language to address uncertainty in self-adaptive systems requirement," *Requir. Eng.*, vol. 15, no. 2, pp. 177–196, 2010.

[18] J. Andersson, R. Lemos, S. Malek, and D. Weyns, "Modeling dimensions of self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems*. Springer-Verlag, 2009, pp. 27–47.

[19] H. J. Goldsby, P. Sawyer, N. Bencomo, D. Hughes, and B. H. Cheng, "Goal-based modeling of dynamically adaptive system requirements," in *15th Annual IEEE International Conference on the Engineering of Computer Based Systems (ECBS)*, 2008.

[20] B. Y. Lim, A. K. Dey, and D. Avrahami, "Why and why not explanations improve the intelligibility of context-aware intelligent systems," in *Proceedings of the 27th international conference on Human factors in computing systems*, ser. CHI '09. New York, NY, USA: ACM, 2009, pp. 2119–2128.

[21] D. Hughes, P. Greenwood, G. Coulson, G. Blair, F. Pappenberger, P. Smith, and K. Beven, "Gridstix:: Supporting flood prediction using embedded hardware and next generation grid middleware," in *4th International Workshop on Mobile Distributed Computing (MDC'06)*, Niagara Falls, USA, 2006.

[22] D. Duggan and F. Bent, "Explaining type inference," in *Science of Computer Programming*, 1995, pp. 37–83.

[23] J. Van Baalen, P. Robinson, M. Lowry, and T. Pressburger, "Explaining synthesized software," in *Proceedings of the 13th IEEE international conference on Automated software engineering*, ser. ASE '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 240–. [Online]. Available: http://dl.acm.org/citation.cfm?id=521138.786885

[24] X. Huang, X. Huang, A. Fiedler, and A. Fiedler, "Proof verbalization as an application of nlg," in *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI*. Morgan Kaufmann, 1997, pp. 965–970.

[25] A. Bundy, "Automated theorem provers: a practical tool for the working mathematician?" *Annals of Mathematics and Artificial Intelligence*, vol. 61, pp. 3–14, 2011, 10.1007/s10472-011-9248-8. [Online]. Available: http://dx.doi.org/10.1007/s10472-011-9248-8

[26] M. Golan and D. R. Hanson, "Duel - a very high-level debugging language." in *USENIX Winter*, 1993, pp. 107–118.

[27] J. Yang, M. L. Soffa, L. Selavo, and K. Whitehouse, "Clairvoyant: a comprehensive source-level debugger for wireless sensor networks," in *Proceedings of the 5th international conference on Embedded networked sensor systems*, ser. SenSys '07. New York, NY, USA: ACM, 2007, pp. 189–203. [Online]. Available: http://doi.acm.org/10.1145/1322263.1322282

[28] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness requirements for adaptive systems," University of Trento, Tech. Rep., 2010.

[29] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 27 2010-oct. 1 2010, pp. 125 –134.

[30] H. Nakagawa, A. Ohsuga, and S. Honiden, "Constructing self-adaptive systems using a kaos model," in *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, oct. 2008, pp. 132 –137.

[31] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," in *SCIENCE OF COMPUTER PROGRAMMING*, 1993, pp. 3–50.

[32] M. M. Lehman and M. M. x Ramil, "Software evolutionŮbackground, theory, practice," *Information Processing Letters*, vol. 88, pp. 33 – 44, 2003.

[33] N. Anquetil, B. Grammel, I. Galvao , J. Noppen, S. Shakil , H. Arboleda, A. Rashid, and A. Garcia, "Traceability for model driven, software product line engineering," in *ECMDA*, 2008.