Engineering Adaptive Requirements

Nauman A. Qureshi
Fondazione Bruno Kessler - IRST
Via Sommarive, 18, 38050
Trento, Italy
qureshi@fbk.eu

Anna Perini
Fondazione Bruno Kessler - IRST
Via Sommarive, 18, 38050
Trento, Italy
perini@fbk.eu

Abstract

Challenges in the engineering of self-adaptive software have been recently discussed and summarized in a seminal research road map. Following it, we focus on requirements engineering issues, with a two-fold, long term objective. The first objective is to support the system analyst to engineer adaptive requirements at requirements-time, the second is to make software able to reason on requirements at run-time in order to enable a goal-oriented adaptation.

Along the first objective, in this position paper we propose a characterization of adaptive requirements. Moreover, we investigate how available techniques aimed at eliciting and specifying domain properties, stakeholders goals and preferences, can provide a practical support to the analyst while capturing adaptive requirements.

1. Introduction

The emerging software technology and the growth of the Internet have spawned the complexity and the maintenance problem for current distributed software applications. Moreover, users require quick and efficient interfacing to the software systems to fulfil their evolving needs.

These problems motivated the proposal of the autonomic computing paradigm [6] that stimulates research to develop software with so called self-* properties, namely self-managing, self-configuring, self-healing, self-optimizing, and self-protecting. A property that is considered transversal to them is *self-adaptivity*, which, according to B. Cheng et al. in [1], can be defined as follows.
Self-adaptive systems can configure and reconfigure themselves, augment their functionality, continually optimize themselves, protect themselves, and recover themselves, while keeping most of their complexity hidden from the user and administrator.

Most of the research works address design-time solutions to provide run-time adaptation [7, 2]. Conversely, requirements engineering for self-adaptive software systems has received less attention so far, as also argued in a recent software engineering research road map for developing *self-adaptive* software systems [1]. Following this research road map, we focus on requirements engineering issues, with a two-fold, long term objective. First of all our objective is to support the system analyst to engineer adaptive requirements at requirements-time. Secondly, we would like to make software able to reason on requirements at run-time in order to enable a goal-oriented adaptation. While starting to address the first objective, in this position paper we propose a characterization of adaptive requirements. By adaptive requirements, we mean that a requirement encompasses the notion of variability in it while elaborating either a functional or quality aspects of the software system. Moreover, we investigate how available techniques for representing domain properties and techniques for modelling stakeholders goals and preferences can provide a practical support to the analyst while capturing adaptive requirements.

The rest of the paper is structured as follows: section 2 recalls basic related works; section 3 proposes a characterisation of the concept of adaptive requirement with the help of a running example; section 4 investigates how available knowledge representation and goal-oriented modelling techniques can provide a practical support to the analyst while capturing adaptive requirements, along the running example. Furthermore section 5 discusses future work and expected results and section 6 concludes.

2. Related Work

A discussion about the nature and the features of requirements for adaptive software can be found in recent work, such as [16, 13]. Whittle et al. in [16], point out the need to capture uncertainty due to the variability in the operational context when specifying requirements for self-adaptive software. They also analyse how natural language can be used to express them using modal verbs.

Salifu et al. in [13] argue that, self-adaptive software needs to monitor, at run-time, the operational context in order to detect requirements violations and to aid the system to switch to (predefined) variants of its behaviours that allows restoring requirements satisfaction. Their work deals with the specification of monitoring requirements and of switching behaviours. We build on the characterization of requirements for adaptive software proposed in the above mentioned works, but differently from [13], we focus on the problem of capturing adaptive requirements that entails monitoring and adaptation bahaviours, a preliminary step to using their approach. Moreover, instead of using natural language or modal logics, as proposed in [16], we aim at investigating the role of goal-oriented modelling languages and knowledge representation techniques (ontologies) to analyse source of variability which motivate the specification of requirements for adaptive software systems. A further motivation of our approach is to provide requirements artefacts that maybe reasoned about by the software at run-time.

Relevant to our approach are also variability design methods based on goal-oriented modelling and research on the use of ontology techniques for requirements engineering. We recall them briefly in the following.

Stakeholders goals have been recognized and used as a useful abstraction to represent the intentional perspective of the users of a system-to-be, providing the rationale behind functional and non-functional requirements of the system, also represented in terms of "system goals" [15, 17]. Goal-oriented requirement engineering approaches, such as [9, 11, 10], exploit goal OR decomposition for analyzing variability in the problem (stakeholders) and in the solution (system) space.

Liaskos et al. in [8] use goal-oriented approach to address the problem of variability by analysing high-level user preferences as goal alternatives and by matching them with the intended system configuration to derive an automatic system configuration. This appears to be a very useful approach to describe the behaviour of self-adaptive software based on goals.

Focusing on self-configuration, Penserini et al. [11] propose to model problem/solution space variability by exploiting goal OR-decomposition, while contribution links to soft-goals, which represent preferences and quality of services, provide the rationale for selecting alternative. An extension to this work by Morandini et al. in [10] is to capture the said adaptivity requirements expressed as goal models that can be mapped to BDI architecture resulting into an agent-based design framework for *self-adaptive* agents. Focusing on the link between requirements- and run-time, goal-oriented approaches provide basis to monitor requirements (violation and refinement) and to reason for run-time behaviour of a system [3].

This motivates our choice to exploit goal-oriented method to model variability, in particular we will refer to the *Tropos* methodology [11].

Ontologies have been recognized as an expressive technique to represent knowledge in a particular domain of interest. Ontologies encompass set of concepts, properties and inference rules expressed using descriptive logic languages such as ontology web language (OWL)[1].

Integration of goal models and ontologies has been recently proposed by Shibaoka et al. in [14]. This approach employs goal-oriented modelling complemented by knowledge representation technique (ontologies) to solve difficulties in refining goal models, thus facilitating the analyst to elicit requirements. This approach makes an important step, thus provides a starting point for our investigation, where we use goal-models and ontologies to elicit requirements for self-adaptive software with the further objective to allow the software system, at run-time, to reason on them.

3. Adaptive Requirements

Requirements for self-adaptive software reflect uncertainty about run-time conditions due to variability in the operational context and in users needs [1, 16].

Software requirements are usually characterised along the functional and non-functional classification. While eliciting and specifying them, the analyst first attempts to characterise the stakeholders' needs that may be elicited through interviews or domain documents, both using natural language, along this classification.

I need a user friendly confirmation message after I book the flight, can be an example of need expressed by the user of a travel booking software service. Here, we can characterise the requirement according to the functional and non-functional perspective, answering questions like, ***What the system should do*** and possibly ***How (well) should it do***. Two functional requirements can be identified (Booking Flight, and Sending Confirmation Message) and one non-functional requirement (User Friendly Message).

Adopting a goal-oriented requirements engineering approach allows us to answer also ***Why the user wants this; why in this way*** questions that result in providing a rationale (i.e. stakeholders goals) behind functional requirements, quality constraints and preferences. Further analysis of software requirements may lead also to answer questions about ***Where*** and ***When*** aspects, leading to a complete specification.

To develop self-adaptive software, we need to make explicit the alternatives in goal achievement, i.e. variability in ***What*** and ***How***, which may be further enhanced by the variability in ***Where*** and ***When*** due to the openness of the operational environment.

[1]http://www.w3.org/TR/owl-features/

This leads to define requirements that are not only *functional* or *non-functional* but also includes *monitoring specification* that take into account the variability in the operational context, *evaluation criteria* and *alternative* software *behaviours* to be adopted at run-time by the software system to ensure the achievement of the intended users' goals. We define such requirements as adaptive requirements, defined in short as: *Requirements that encompass the notion of variability associated to either a functionality or a system quality constraint.*

3.1. Running Example

We refer to a scenario from the tourism/travel domain to help clarifying how the analyst can capture variability and flexibility that identify adaptive requirements.

Mr. John is a business professional and travels frequently around the world for his business reasons. He uses his laptop and his PDA to manage his meeting and office work, while travelling. He usually books his travel tickets and hotel accommodation in advance using available internet applications and web-services, but many times he needs an urgent booking, which needs a lot of last minute search using internet. Moreover, he often encounters problems due to not being informed timely about delays in checking in for flights and cancelled flights etc. This puts him in a situation to bear expensive alternative solutions. In addition, the software should offer travel and leisure trips information, which should be readily available on his device based on his travel.

Mr. John needs a software application, which can manage his booking (Itinerary) by monitoring his given preference (booking reference or schedule etc.) and informing him about the changes well before. We will call it Travel Comp from now on. Let's pick up a simple user need as that of being informed about confirmed booking. The analyst may describe it with the following statement and then identify from it some requirements for the Travel Comp software application. *A user friendly confirmation message, after booking is processed, must be communicated to the user on his current device with proper representation.*

In this example, there are four functional requirements namely book a ticket, send confirmation message, message communication to device and format representation of message (addressing ***What*** is required). Whereas one non-functional requirement i.e. user friendly message (addressing ***How*** well it should send). The analyst can now make explicit the variability in user's devices (implicit in the above requirement statement) and re-phrase it as follows:
A user friendly confirmation message, after booking is processed, shall be communicated to the user on his current device (e.g. PDA/Laptop) by seamlessly observing (monitor) the user's context (Profile, Location, Device), in order to deliver required/personalized contents to his current device i.e. PDA/Laptop.

The variability in user's device (PDA/Laptop) demands monitoring at run-time to make the software aware of the current operational context conditions and able to send the confirmation message in a proper (formatted) way to the user's current device, so meeting the original user goals.

Unanticipated events, which may occur with significant probability, should also be considered, such as message is not delivered correctly or connection lost. This calls for adding requirements to the software to make it degrading gracefully or involving the user to take corrective actions. To cope with such level of uncertainty (what might happen), we also need to ensure at requirements-time that apart from addressing previous questions the requirements shall also try to address ***When*** and ***Where*** uncertainty. The above example can be re-stated as follows:
A confirmation message for booking is generated as soon the booking is processed, and required to possibly communicate the message to the user on his current device (e.g. PDA/Laptop) by seamlessly observing (monitor) the user's context (Profile, Location, Device), run-time events and QoS attributes until the message is delivered in a correct format (by scaling it, size, etc) and with personalized representation (e.g. SMS, Email) to his current device i.e. PDA or a different way to notification is applied.

This allows to address situations in which, at run-time, the message could not be confirmed as delivered either due to wrong message delivery or connection lost. Then, using for instance users' contact info, the software can either send an email to his secretary or notify his friends.

In the above example, it is not explicitly made precise when the message will be delivered or it must be delivered. That is, flexibility in ***When*** aspects is considered. Moreover, variability in users' devices (***Where***), users' preferences (goals - ***What, Why***), and uncertainty in the operational context is considered, motivating the explicitation of different behaviours (plans - ***How***), to achieve the user' s original need.

4. Capturing Adaptive Requirements

The overall idea is, on one hand, to elicit stakeholders' needs and to analyze them in terms of the alternative ways to meet them, by exploiting goal-oriented methods. On the other hand, to exploit domain ontologies to express application domain and operational context assumptions, as well as to capture variability in them. This domain ontology and the goal models are linked together to capture the essential aspects of the requirements for the adaptive system. This is depicted in fig. 1, with reference to the "Travel Comp"software application. In the right side, an excerpt of goal-model illustrates the goals of the software system, ex-

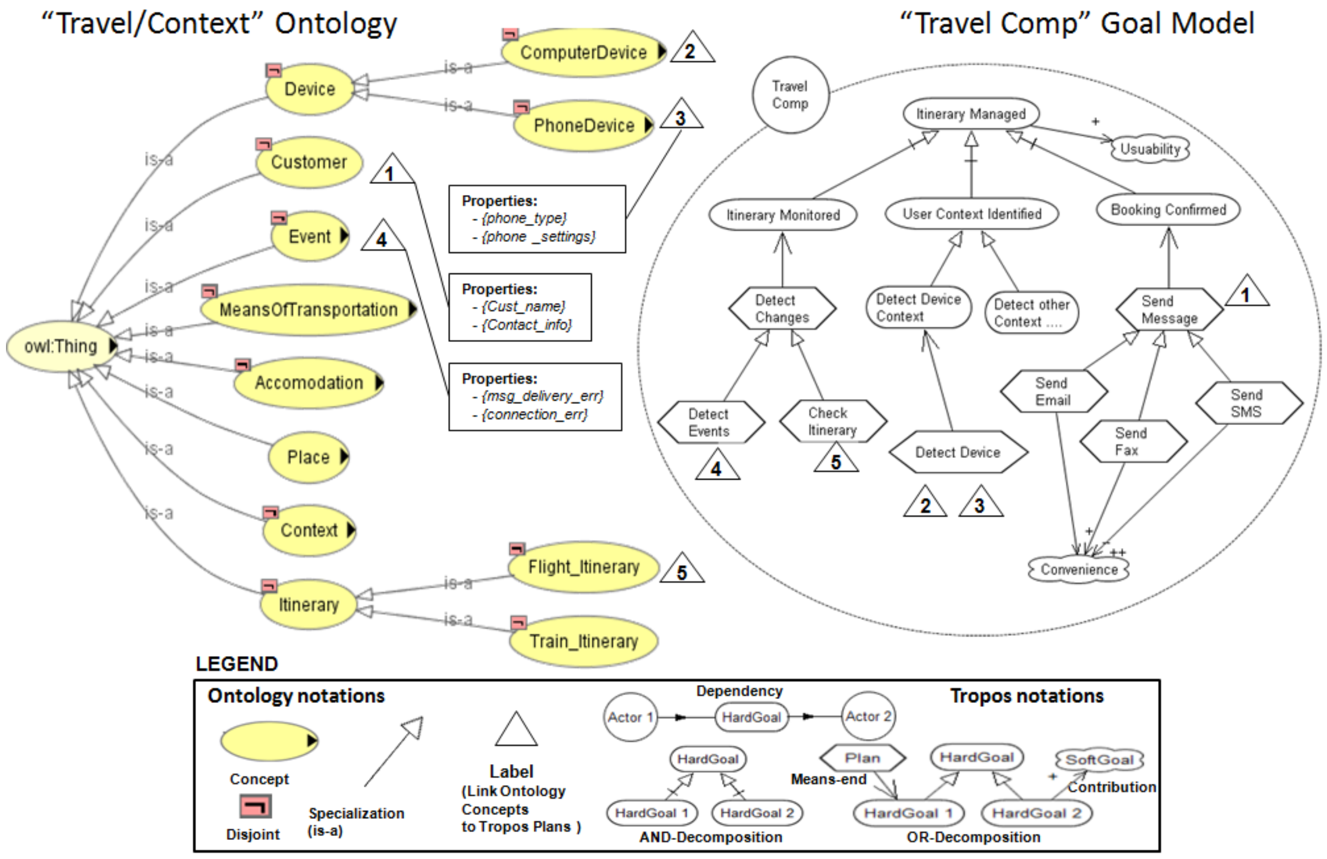

Figure 1. Annotating Goal Model with domain properties

pressed in *Tropos* notation, at the left side, an example of the domain ontology is shown.

The resulting annotated goal model helps the analyst to reason about the alternative behaviours of the system, which takes into account variability in the application domain and in the operational context. Moreover, linking the concept properties, from ontology to goal model, facilitates to derive monitoring specifications and evaluation criteria to drive the switching among variants of behaviour at run-time, so following the definition of adaptive requirements given in the previous section.

Goal Modelling. Following the *Tropos* software development methodology [11], we move from a goal-oriented model of the domain's stakeholders (including users of the system-to-be) to the specification of the system requirements by delegating users goals to the system. Hard-goals represent the rationale behind functional requirements (depicted as ovals in fig. 1, right side) and soft-goals (cloudy shape) represent non-functional aspects of the system-to-be.

Elaborating the example introduced in the previous section, we analyse the stated requirement in terms of hard-goals and soft-goals thus addressing the (***Why***) question along with the (***What***).

The goal Booking Confirmed has been delegated by the user (Mr. John in our scenario) to the system, represented by the actor "Travel Comp", and it is analysed as a sub-goal of a more general goal of the system, namely Itinerary Managed, through AND-decomposition.

The other sub-goals, i.e. Itinerary Monitored and User Context Identified, corresponds to the user needs of being supported by the software system also while traveling, so being in a situation in which context and devices may flexibly change. The analysis of these two additional goals becomes essential to end up with a specification suitable for an adaptive system, that is including monitoring conditions and evaluation criteria for switching behaviour.

Following the details of the Booking Confirmed analysis, we may found that this goal is achieved through a plan $\langle SendMessage \rangle$, depicted in terms of means-end relation in fig. 1 [2]. This plan specifies the action of sending a confirmation message to the user device as soon the booking is processed. This plan can be realized by alternative behaviuors, namely $\langle SendEmail \rangle$, $\langle SendSMS \rangle$ and

[2]The "Travel Comp" analysis is is not fully described here for space reasons, we remind the interested reader to [12] for further details. So, for instance, it is not reported here how the ticket booking service is realized.

```
//$Plan\_Model(\langle SendMessage \rangle)$ to accomplish Goal (BookingConfirmed)
begin procedure $Plan\_Model(\langle SendMessage \rangle)$
do triggerGoals [UserContextIdentified,ItineraryManaged];
    begin
        for Goal [UserContextIdentified]
        do executePlan $\langle DetectDevice \rangle$; //@param: phone_type, phone_setting
        return; //@result: device
    end;
    begin
        for Goal [ItineraryManaged]
        do executePlan $\langle DetectChanges \rangle$; //@param: msg_delivery_err, conn_err
        return;//@result: eventMessage
    end;
$decision = decision\_on\_AltPlans(device, eventMessage)$;
case $decision$:
        - Select case:$\langle SendSMS \rangle$; //if device = PDA, eventMessage = null
        - Select case:$\langle SendEmail \rangle$; //if device = Laptop, eventMessage = null
        - default case: $\langle SendFax \rangle$; //if device = null, eventMessage = null
$ifnot$ [$decision$]
    $then$ lookupContact; //@param: cust_name, contact_info
$alt\_decision = decision\_on\_AltPlans(cust\_name, contact\_info)$;
case $alt\_decision$:
        - Select case: $\langle SendEmail \rangle$; //contact_info
        - default case: $\langle SendFax \rangle$; //contact_info
end procedure;
```

Figure 2. Example of Plan description

$\langle SendFax \rangle$, represented by the plan OR-decomposition. These three alternative plans address the variability in the operational context, which affect (***How***) the goal of confirming booking can be achieved by the software system. Criteria for selecting the right alternative are represented by contribution relationships (arrows labeled with "+" or "-") to soft-goals. For instance, the selection of the plan $\langle SendSMS \rangle$ should be performed when the user's device is a PDA, as it contributes (++) positively to the users' soft-goal {Convenience} as compared to $\langle SendFax \rangle$, which contributes (–) negatively to it.

Information needed to drive the plan selection are collected at run-time by the system following monitoring requirements expressed in the analysis of the goals Itinerary Monitored and User Context Identified. For instance the plan $\langle DetectEvent \rangle$ that is part of the means-end analysis of the Itinerary Monitored goal contains specification of how to detect unforeseen events, as message delivery error or connection lost, and provide criteria for the system to switch to a behaviour in which Mr. John contact info are used to inform colleagues or friends about the confirmed booking.

Domain Modelling. Representing the knowledge about the domain using ontology provides a way to elicit domain assumptions. We model application domain and operational context properties in the ontology as shown on the left side of the fig. 1. For example, in the ontology we identify the common concepts used in travel domain such as Itinerary, Accommodation, Means of transportation (by plane or by train etc.). After identifying the concepts, we define relationships between these concepts. It helps in understanding the said domain assumptions that remains true in a domain of interest. To express these assumptions we use concept properties and inference rules, which represents them. For example, one possible domain assumption can be: *Customer's itinerary is valid not before and not after the departure date.* Moreover, concepts that describe the operational context are represented in our ontology, such as Context, Device, Event having a property *(Message deliver error)* and Customer having a concept property such as *(Contact Info)* etc.

Linking Ontology to Goal Model. The links between domain concepts and their properties represented in the ontology to the plans and goals represented in the goal model are shown in the fig. 1 (see label 1 to 5). For instance, the concept Phone Device in the ontology is linked with the plan Detect Device in the goal model (with the label 2 and 3). These labels associate the properties *phone_type* (Mobile Phone, PDA etc.) and *phone_setting* (e.g. Operating system, memory etc.) with the plan Detect Device to provide an additional information to the plan's specification. Similarly, the concept Event is associated with plan Detect Events, which provides the plan to detect events such as *msg_delivery_err* as shown in the fig. 1 as label 4.

This helps analysts in detailing software behaviours that encompass not only the actions to satisfy (functional) goals, but also monitoring and evaluation actions. Fig. 2 describes the plan $\langle SendMessage \rangle$, which provides the means to achieve the (functional) goal BookingConfirmed. This plan requires to trigger monitoring goals that are pursued in parallel, providing data for the evaluation actions that drive the selection of the right alternative e.g.$\langle SendSMS \rangle$ to send confirmation message to the users' device (assuming the device is identified as PDA).

5. Discussion

The requirements specification problem has been recently redefined in [5] in terms of a planning problem, aiming at specifying what a software system should do to meet users goals, quality constraints and preferences, under a given set of domain assumptions.

In our research we are applying this definition to the case of requirements for adaptive software with the aim to propose a methodology for the analyst to specify these requirements. Here the analysis of the sources for variability plays a crucial role. First variability in the application domain and the operational context needs to be identified, second the alternative software behaviours, that may take place in the different context conditions, to achieve users goals and preferences have to be found. A basic assumption underlying our work is that users goals are stable (e.g. I want to receive confirmation after booking has been processed) since they relate to concrete needs, while the context in which the user wants these goals to be achieved may vary (i.e. moving with a PDA, using his laptop, etc.). User preferences (e.g. I prefer SMSs) and quality constraints may also be relaxed to allow for gracefully degrading behaviours.

In this position paper we are investigating the usefulness of knowledge representation and modelling techniques to support the analysis of this variability, and to represent variants of software behaviours which assume context monitoring and evaluation steps that allow to realize an adaptive system. Among the reasons for using ontologies, also the fact that a number of domain ontologies are available for reuse on the Internet, [3] so mitigating the risk of an additional modelling effort when performing requirements analysis. Concerning goal-oriented modelling, considerable work is showing its effectiveness as a method for high variability design. In addition, existing versions of goal-oriented modeling language using first-order linear time temporal logic may provide more expressive way to represent temporal information [4].

Further work is required to consolidate this analysis, especially with real scenarios, and to define a step by step analysis process.

The role of these requirements artefacts (goal models plus ontology) to enable adaptation at run-time needs also additional investigation. A related work, along this direction, can be considered the approach proposed by Morandini et al. [10] in which goal models including variants of behaviour are implemented into BDI agents. Differently, in our work, we would like to address other implementation platforms and middleware based architectures.

6. Conclusions

In this position paper, we presented a characterisation of requirements for adaptive systems. Moreover, we investigated the usefulness of the knowledge representation techniques and goal-oriented modelling to support the analysis of variability, a key step towards understanding the adaptive requirements for self-adaptive software systems.

References

[1] B. H. C. Cheng, H. Giese, P. Inverardi, J. Magee, and R. de Lemos. Software engineering for self-adaptive systems: A research road map. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 08031 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[2] S.-W. Cheng, A.-C. Huang, D. Garlan, B. R. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. In *ICAC*, pages 276–277. IEEE Computer Society, 2004.

[3] A useful ontology search service is available at: http://swoogle.umbc.edu

[3] M. S. Feather, S. Fickas, A. V. Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *IWSSD '98: Proceedings of the 9th international workshop on Software specification and design*, page 50, Washington, DC, USA, 1998. IEEE Computer Society.

[4] A. Fuxman, L. Liu, J. Mylopoulos, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in tropos. *Requir. Eng.*, 9(2):132–150, 2004.

[5] I. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the core ontology and problem in requirements engineering. *International Requirements Engineering, 2008. RE '08. 16th IEEE*, pages 71–80, Sept. 2008.

[6] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[7] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. *Future of Software Engineering, 2007. FOSE '07*, pages 259–268, May 2007.

[8] S. Liaskos, A. Lapouchnian, Y. Wang, Y. Yu, and S. M. Easterbrook. Configuring common personal software: a requirements-driven approach. In *RE*, pages 9–18. IEEE Computer Society, 2005.

[9] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*. IEEE Computer Society, September 2006.

[10] M. Morandini, L. Penserini, and A. Perini. Towards goal-oriented development of self-adaptive systems. In *Proceedings of (SEAMS '08)Workshop on Software engineering for adaptive and self-managing systems*, pages 9–16. ACM, 2008.

[11] L. Penserini, A. Perini, A. Susi, and J. Mylopoulos. High variability design for software agents: Extending Tropos. *TAAS*, 2(4), 2007.

[12] N. A. Qureshi and A. Perini. Engineering adaptive requirements. Technical report, 2008.

[13] M. Salifu, Y. Yu, and B. Nuseibeh. Specifying monitoring and switching problems in context. *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 211–220, Oct. 2007.

[14] M. Shibaoka, H. Kaiya, and M. Saeki. GOORE: Goal-Oriented and Ontology Driven Requirements Elicitation Method. In *Advances in Conceptual Modeling - Foundations and Applications*, pages 225–234, Auckland, New Zealand, Nov. 2007. Springer. LNCS 4802.

[15] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *5th IEEE International Symposium on Requirements Engineering (RE)*, page 249. IEEE Computer Society, 2001.

[16] J. Whittle, P. Sawyer, N. Bencomo, and B. H. Chen. Reassessing Languages for Requirements Engineering of Self-Adaptive Systems. In *4th International Workshop on Service-Oriented Computing Consequences for Engineering Requirements (SOCCER'08)*, 2008.

[17] E. Yu and J. Mylopoulos. Understanding why in software process modelling, analysis, and design. *Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on*, pages 159–168, May 1994.