REFAS: A PLE Approach for Simulation of Self-Adaptive Systems Requirements

Juan C Muñoz-Fernández, Gabriel Tamura, Irina Raicu, Raúl Mazo, Camille Salinesi

REFAS: A PLE Approach for Simulation of Self-Adaptive Systems Requirements

Juan C. Muñoz-Fernández
CRI, Université Paris 1 Panthéon Sorbonne, Paris, France
Universidad Icesi, Depto. de TIC, Cali, Colombia
jcmunoz@icesi.edu.co

Gabriel Tamura
I2T/DRISO Research Group
Universidad Icesi, Depto. de TIC, Cali, Colombia
gtamura@icesi.edu.co

Irina Raicu
CRI, Université Paris 1 Panthéon Sorbonne, Paris, France
Bucharest University of Economic Studies, Bucharest, Romania
irina.raicu@malix.univ-paris1.fr

Raúl Mazo
CRI, Université Paris 1 Panthéon Sorbonne, Paris, France
raul.mazo@univ-paris1.fr

Camille Salinesi
CRI, Université Paris 1 Panthéon Sorbonne, Paris, France
camille.salinesi@univ-paris1.fr

ABSTRACT

Model simulation has demonstrated its usefulness in evaluation and decision-making for improving preliminary versions of artefacts before production. Particularly, one of the main goals of simulation is to verify model properties based on data collected from its execution. In this paper, we present the simulation capabilities of our REFAS framework for specifying requirements models for dynamic software products lines and self-adaptive systems. The simulation is controlled by a feedback loop and a reasoning engine that operates on the functional and non-functional requirements. The paper contribution is threefold. First, REFAS allows developers to evaluate and improve requirements models through their simulation capabilities. Second, REFAS provides rich feedback in its interactive simulations for the human modeller to make informed decisions to improve her model. Third, REFAS automates the generation of simulation scenarios required to verify the model adequacy and correctness. We evaluate our contribution by comparing the application of REFAS to a case study used in other approaches.

General Terms

Algorithms, Documentation, Design, Languages

Keywords

Requirements engineering, dynamic software product lines, dynamic adaptation, simulation, MAPE-K loops.

1. INTRODUCTION

Self-adaptive software (SAS) systems automatically adjust their behaviour in response to changes in the surrounding context in which they are executed. Dynamic software product line (DSPL) [1] engineering intends to produce software that can be adapted at runtime and in this sense they are a particular case of SAS. Realizing this self-adaptive ability implies to cope with the inherent uncertainty that execution contexts pose for this kind of systems, which is certainly one of the most difficult aspects to specify and control. For instance, different contexts may demand different trade-offs in requirements, and unanticipated contexts may even lead to entirely new requirements. Thus, verifying the adequacy of these requirements, specified as a product line model, with respect to context uncertainty constitutes a difficult and time-consuming goal, as it implies to check its behaviour in several configurations under changing contexts of execution. In this setting, interactive simulation is a useful tool to explore the system's response and determine incorrect or unexpected behaviour, by allowing the modeller to expose the system to uncertainties actually discovered by analysing the simulated status of the system.

Languages and frameworks for modelling requirements usually focus on requirements specification and automated synthesis of a corresponding configuration, offering little support for modelling changes in the context of system execution. In a previous work, we proposed a requirements engineering framework for SAS (REFAS) and its corresponding modelling language [2]. This framework aims to address context uncertainty and to be sufficiently expressive for SAS requirements. However, as in other approaches, after capturing the context-dependent requirements, their validation is a critical next step not enforced to be performed systematically. Therefore, a *first challenge* is to address how to realize simulations that can be performed based on different views used to represent the variability of SAS requirements, each supporting different kinds of information and addressing different concerns. A *second challenge* is to enable the simulations to provide feedback in intermediate execution states and contextual information for the modeller to evaluate, correct, and complete the requirements model, before continuing with the development phase. Finally, *a third challenge* is to provide the simulation framework with automated tools for specifying and generating relevant scenarios to help verify the model adequacy and correctness of SAS requirements represented as a product line model.

In this paper, we present the simulation capabilities for our SAS and DSPL requirements framework as a means for validating them.

SPLC 2015, July 20 - 24, 2015, Nashville, TN, USA
 ISBN 978-1-4503-3613-0/15/07…$15.00
DOI: http://dx.doi.org/10.1145/2791060.2791102

For the simulation, the framework includes a language for specifying an inventory of assets in the form of software components, and a reasoning engine to compose them in order to satisfy the changing requirements. We describe a prototype implementation of our simulation approach, and illustrate our contribution by comparing the application of our framework to a previously published case study.

The remainder of this paper is organized as follows. In Section 2 we present the motivation. In Section 3 we summarize the REFAS framework and its simulation capabilities. In Section 4 we present the evaluation results. In Section 5 we discuss related work, and we conclude and discuss future work in Section 6.

2. MOTIVATION

2.1 RUNNING EXAMPLE

In this paper, we use the GridStix case study, which has been used previously in other approaches (e.g., [4] [5] [6]). GridStix describes the problem of a river highly prone to flood the lands on a remote rural area. Alerts about probable floods help to mitigate human and material losses. Nonetheless, false alerts imply critical but unnecessary costs of transportation and other life- and value-preserving activities. Therefore, the accuracy of alerts is a critical factor for solving this problem. GridStix requires a network of wireless sensors monitoring the river flood, connected to very small data processors. These processors compute the flooding probability based on historic information and sends respective alerts through wireless protocols. Of course, this battery-operated infrastructure, located in a remote rural area, requires energy optimization to prolong its operation, under changing conditions of execution, while preserving the accuracy of alerts.

Context conditions of execution, such as weather and season, imply critical variables to consider given their effect in different system aspects, such as battery-life and flooding probability. Different context conditions imply different system configurations in terms of software components, which in turn imply different levels of power and memory consumption. All of the aforementioned requirements are characteristic of dynamically reconfigurable (i.e., self-adaptive) software systems. In this paper, however, we focus on three factors of context dynamism also included in other approaches. First, communication between data processors can use Wi-Fi or Bluetooth. For remote monitoring and alert notification, another option is GPRS/GSM. Wi-Fi offers lower latency and is more robust than Bluetooth at the expense of higher power consumption [4]. Second, for data transmission between data processors, two strategies of routing can be used: shortest path and fewest hops. The first consumes less power but offers less performance than the second [4]. Third, for performing preliminary analysis on the speed of flood water, data processors can process images of the river using centralized or distributed algorithms [6]. Distributed algorithms can improve the analysis results at the expense of higher power consumption [4].

2.2 Simulation of Dynamically Changing Software Requirements

SAS requirements models must consider not only context-dependent variables and conditions (e.g., weather, season and geographical location), but also constraints imposed by the problem for its solution (e.g., technical and geographical limitations on the power sources), in addition to the usual functional and non-functional requirements. Thus, to preserve the satisfaction of changing requirements, the running system must reconfigure itself at execution time. However, languages and frameworks for modelling requirements usually focus on requirements specification and automated synthesis of a corresponding solution, but not on simulation. Therefore, a first challenge to address is to realize simulations that can be performed based on different types of views supporting different kinds of information for different concerns and perspectives. These different types of views, ideally defined by the user herself, would provide complementary information for the simulations to be more accurate, and for the modeller to make better-informed decisions to improve the whole requirements model.

A second challenge is to enable the simulations to provide rich feedback in terms of simulation states and contextual information for the human modeller to evaluate, correct, and complete the requirements model, before continuing with the development phase. This information should be discoverable at simulation time, by direct interaction with, and inspection of the intermediate simulation states. Finally, a third challenge is to provide the simulation framework with tools for specifying and automatically generating relevant scenarios to help verify the model adequacy and correctness. These scenarios, usually hand-coded in time-consuming and error-prone tasks, must capture the diversity of context situations the system can face at execution time.

3. REFAS: Simulation of SAS Requirements Models

In this section, we present our Requirements Engineering For (Self)-Adaptive Systems (REFAS) framework and its capabilities for simulating requirements models. Nonetheless, we need to introduce first how we define requirements models with REFAS.

3.1 REFAS Concepts and Views

To realize simulations based on different types of views supporting different concerns and perspectives, addressing our first challenge, we designed a generic requirements meta-modelling language [2]. This language defines basic concepts and relations for defining requirements and allows the modeller to define arbitrary types of views by combining these concepts and relations. These arbitrary types of views specify different concerns of the requirements model and provide complementary information that can be used for providing more accurate simulations.

REFAS provides nine concepts for building requirements models.

Goals represent high-level functional purposes that the system must achieve, whereas *soft goals (SG)* represent non-functional requirements (e.g., QoS levels) that the system should satisfy according to context conditions of execution. A (goal) *operationalization* represents a way to satisfy a goal (e.g., through a software component). However, this satisfaction is conditioned to the validity of *assumptions*. That is, an assumption represents the conditions under which an operationalization confidently satisfies a goal. A *claim* express the SG expected level of satisficing by a given operationalization. A *Variable* represents the current value of a particular variable of interest of the system's execution state. Variables can be grouped in a *ConcernLevel*. A *soft dependency* specifies the required level of a soft goal satisficing, under a given context situation. Finally, we adopt *Features* exactly in the sense of feature models (FM).

Our modelling language specifies five views:

The soft goals view supports the soft goals definition and the relations between them. For our running example, GridStix, the soft goals are *FaultTolerance* (Fig. 3-label A*)*, *EnergyEfficiency* (Fig. 3-label B) and *PredictionAccuracy* (Fig. 3-label C).

The goal/operationalization variability view represents the variability and their inter-relations. This view optionally includes restrictions on the variability satisfaction in terms of assumptions. The variability view can also support the use of features instead of goals and operationalizations. Figure 1 presents goals defined for GridStix. Three of the goals have two operationalizations each. The operationalizations are mutually exclusive, and all the goals are required. For instance, GridStix specifies two possible but mutual exclusive operationalizations for the goal *CalculateFlowRate*.

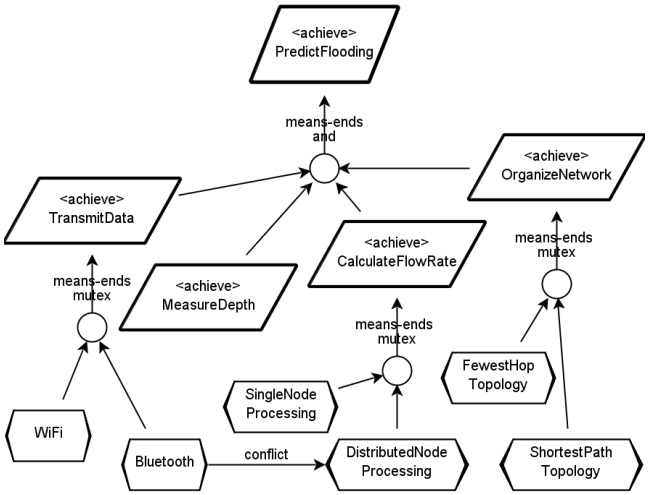

Figure 1 Goal/Operationalization Variability View of GridStix

The Context view defines the relevant context variables on which possible system adaptations depend upon. The values of these variables may require different levels of satisfaction for soft goals. Figure 2 presents the context view of GridStix with two Boolean variables (*FloodPredicted* and *HighFlow*) and one enumeration variable (*BatteryState*) with two valid values (low, high).

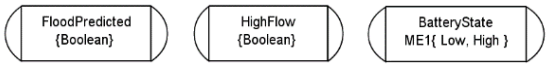

Figure 2 Context View of GridStix

The soft goals satisficing view represents the conditional relations between the goal/operationalization variability view, the soft goals view and the context view. They are expressed with soft dependencies, and claims added with constraints. The constraints can combine numeric and Boolean expressions to define their conditions of activation for both types of conditional relations. Figure 3 presents the soft goals satisficing view with seven claims (cf. CL in the figure) and four soft dependencies (cf. SD in the figure). We explain a conditional relation as follows. CL4 constrains the expected level of the *EnergyEfficiency* soft goal to 4 if the system uses the *Bluetooth* and *SingleNodeProcessing*.

The assets view represents the implementation components of the system. This view defines the assets, their relations and maps each operationalization (or adaptation features) to software components.

3.2 The Simulation Control Loop

Our requirements model attempts to capture as completely as possible the variability of SASs by means of soft goals, foreseeable context conditions and required corresponding system adaptations, constraints and their inter-dependencies. In this section, we complete the core semantics with the behavioural semantics, that is, the meaning of the requirements model at simulation time.

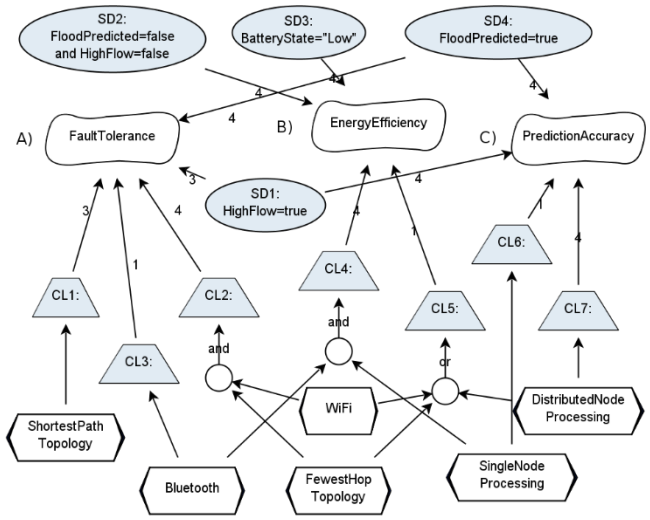

Figure 3 Soft Goal Satisficing View of GridStix

We define a simulation as a sequence of scenarios where a scenario is the definition of a partial mapping of the model's context variables to corresponding values. Thus, running a simulation means to execute the constraint program (i.e., the core semantics of the requirements model) with each of the simulation scenarios (i.e., a constraint-satisfaction problem to find a configuration to satisfy the context-dependent requirements) in an interactive sequence.

Therefore, to realize this interactive sequence we use a simulation control loop implementing the Monitor-Analyser-Planner-Executor-Knowledge base (MAPE-K) reference model [7].

Monitor. The monitor goal is to identify and report internal and external context events. In REFAS, there are two monitored sources of events. First, the requirements model defines the concepts, its attributes, and relations. For example, an attribute identifies whether the concept is in the model. Second, the requirements model configuration that defines restrictions on the selection and exclusion of concepts, and also the values for some of the variables.

Analyser. The analyser evaluates the events notified by the monitor and the simulation's current configuration state, as specified by the requirements model. The simulation's current configuration results from the aggregation of the requirements model design and configuration, and the simulation configuration. The analyser invokes the planner if the configuration is not optimal or invalid.

Planner. The planner evaluates the current configuration state and computes a new configuration by invoking the obtain solutions method, logging the results to save the configuration. The planner notifies the executor with the configuration plan (i.e., a configuration solution) and the analytical execution information.

Executor. The executor formats the configuration and variable values of the selected solution using JavaScript Object Notation (JSON), writes the output files and triggers updates on the user interface. The user interface includes the requirements model, the dashboard, the statistical information, and alerts in case of error.

Knowledge base. The knowledge-base element is a data structure storing the set of constraints automatically generated from the concepts and relations between all the requirements model views. This element also contains the constraints created with the values of variables used in conditional expressions of soft dependencies and claims. The constraints are used by the analyser and planner.

3.3 Simulation Controller and Generator

To provide appropriate and interactive feedback for the modeller, thus addressing our second challenge, the simulator must deploy

appropriate controls, display useful results and be fed with correct data inputs. The coordination of these actions is performed by the simulation controller, according to the modeller's simulation needs.

The simulation's controller interface provides a panel for specifying and modifying all of the concepts comprising the user's requirements models in addition to the complementary functions including simulation log, among others. The simulation requires two configuration files. First, the file which may include initial values for concept attributes and variables. The variables are of two types: external context and target system. Second, simulation parameters defines the timing for simulation, the initial simulation configuration, the type of concepts to consider in the simulation, the folders for storing simulation files, and the type of simulation constitutes the simulation parameters of this element.

The *simulation controller and generator* element automate the generation of simulation scenarios. The inputs for this configuration include random values for the requirements model's concepts and variables. The user can combine the alternatives to adjust the simulation inputs to the aspects she wants to evaluate.

A scenario defines a combination of values for external context and target system variables that require a particular satisficing level over at least one soft goal of the requirements model. The combination of all those values is evaluated to generate the scenarios. To reduce the explosion in the number of scenarios, the soft goals, and other concepts satisfaction/selection may be maximized/minimized in the requirements model.

3.4 Simulation Visualization

The simulation graphical user interface provides simulation process feedback to address our second challenge. The GUI provides two perspectives, one for the modelling, and one for simulation. The first is used to define the concepts and views of the requirements model, whereas the second is used entirely for performing its simulation and providing useful information for the modeller to evaluate the correctness and adequacy of the requirements model.
In the graph, elements selection is represented by a rectangle on top of the element. The rectangle colour alternatives are: green for a selected element; red for a not selectable element; and non-colour for a selectable element. Moreover, in the configuration/simulation perspective, the rectangle has three circles. Circles from left to right represent the selection at design time, configuration time and simulation time with the same colours as the rectangles.
The dashboard summarizes the selection of concepts to explore the configuration's relevant concepts. SG and variables include its value. A dashboard for GridStix is presented in Fig. 4-label (B).

The statistical information provides information about the number of solutions, selected concepts, soft goals satisficed, and activated claims and soft dependencies. Statistical information includes the execution time of the last iteration, the solver, the compilation and the total. Some execution times are illustrated in Fig. 4-label (C).

The simulation detects different situations during the simulation and notifies the user accordingly. For example, it notifies about problems with the requirements model design, the definition of conditional expressions, or no solution found for a particular combination of context variables.

4. EVALUATION

To evaluate our framework, we implemented the REFAS framework in a software tool that we named VariaMos [8].

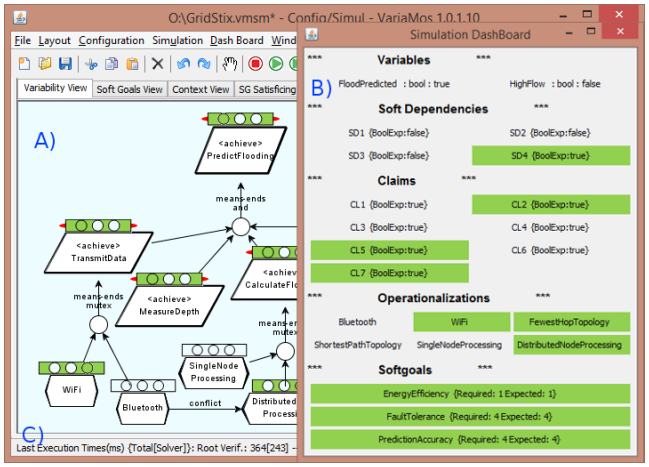

Figure 4 Model configuration/simulation perspective of GridStix. VariaMos screenshot with dashboard

To illustrate the simulation of context changes, we explain an adaptation with a specific context and configuration of the GridStix requirements model. An initial GridStix model configuration selects *ShortestPathTopology*, *SingleNode Processing* and *Bluetooth* operationalizations and the variables *FloodPredicted* and *HighFlow* in *false* as shown in Fig. 4-(B). A rise in the river flow is simulated with a scenario setting the variable *HighFlow* to *true*. The monitor identifies the change in this variable and calls the analyser; the change has no valid configuration because *FaultTolerance* is non-negotiable, and its required level cannot be satisfied. Then, the planner tries to identify an alternative configuration. As a result, the new configuration adapts from *Bluetooth* to *Wi-Fi* and from *SingleNodeProcessing* to *DistributedNodeProcessing*.

REFAS with VariaMos satisfies our first challenge by supporting different types of views and complementary information to obtain more accurate simulations and help the modeller to make better-informed decisions to improve the SAS requirements. Regarding the second challenge, VariaMos provides feedback in terms of execution state and contextual information for the human modeller. The feedback helps to evaluate and correct the SAS requirements. The user can modify the next iteration, according to the feedback analysis. Finally, REFAS/VariaMos solves the third challenge by avoiding the hand-coded development of scenarios. The simulation framework automatically generates relevant scenarios to verify the requirements adequacy and correctness. This generation represents an improvement over other approaches, such as Genie [5].

We consider the simulation of SAS important due to the difficulty of testing SAS in several configurations scenarios. The simulation performed before or during development, and during the execution provides different benefits. The former identifies errors or not considered conditions of the definition of the system and perform adjustments before the complete development. The latter supports the system maintenance in terms of corrective, adaptive and perfective maintenance.

5. RELATED WORK

The work presented in this paper has been influenced by other approaches for simulation or execution of software systems.

Sawyer et al. [3] proposed a goal approach for requirements modelling that requires the manual transformation of the resulting model to a constraint program to execute in the first release of VariaMos [9]. The new VariaMos [8] provides an automatic calculation of core concepts, error verification, configuration and

simulation of the requirements models. Our approach also supports complex Boolean and numerical expressions for soft dependencies and claims. This increases the expressivity of the reasoning to support the dynamic adaptation representation.

Genie [5] proposed a component-based approach to deal with architectural adaptations. Genie defines the scenarios for variability based on transition diagrams. We consider this well suited for small systems. However, in the case of bigger self-adaptive systems, scenarios should be derived from the constraints defined for the system, including variable values and transition constraints.

Specification Animator [10] supports various agents interacting to construct the behaviour of components of a system and their environment. Our approach is centred on the designer, supporting the modelling and the simulation of the requirements models. We can evaluate the validity of the system according to scenarios but not managing stakeholder decisions within the model.

CAMP [11] proposed an abstract layer architecture, integrating concerns from context-aware and self-adapting systems. They focus on the solution space, including the analysis and decisions defined explicitly by rules on composites. We cover the problem space (goals and operationalizations) and the solution space (reusable domain components).

DiVA [12] supports four meta-models: DSPL, context, reasoning and architecture. From the meta-models, DiVA offers testing for early validation and simulation. The simulation does not include system or adaptation interactions.

FUSION [13] supports adaptation of systems from feature models and defines utility functions to measure the satisfaction of functional and QoS objectives. FUSION focuses on the discovery of relations between the features and the metrics implementing learning algorithms. FUSION targets the execution of systems, not the simulation. It also requires all the features to resolve issues to be preconceived.

6. CONCLUSIONS AND FUTURE WORK

We have presented REFAS, our PLE approach supporting the modelling, configuration and simulation of SAS integrating a MAPE-K loop. The benefit of REFAS for simulation is threefold.

First, we support the designer in the modelling SAS requirements with our graphical language. The requirements model is automatically transformed into a constraint program and exploit for simulation. Second, we provide feedback in terms of execution state and contextual information for the designer. The feedback functionality is integrated into VariaMos. From VariaMos, the designer can simulate before the development or implementation of the target system. Third, our approach avoids the hand-coded development of scenarios, by automatically generating relevant scenarios to verify the model adequacy and correctness.

We are interested in extending our framework in three main directions. First, to implement the DYNAMICO [14] reference model to support the adaptation at the goal and context levels. DYNAMICO proposes a clear way to separate the three levels of dynamics of context-driven self-adaptive systems. Second, to support multiple instances of the dynamic adaptation level. We plan to experiment with the decentralized reasoning of those multiple instances adapting the VariaMos implementation. Third, we will incorporate temporal logic and transition support. We are working to support actions for each transition.

7. ACKNOWLEDGMENTS

This work was supported in part by grant 0369-2013 (project SHIFT 2117-569-33721) from the Colombian Administrative Department of Science, Technology and Innovation (Colciencias).

8. REFERENCES

[1] S. Hallsteinsen, M. Hinchey, P. Sooyong, and K. Schmid, “Dynamic Software Product Lines,” IEEE Computer, 41 (4), pp. 93-95. 2008

[2] J. C. Munoz-Fernandez, G. Tamura, R. Mazo, and C. Salinesi, "Towards a requirements specification multi-view framework for self-adaptive systems," *Computing Conference (CLEI), 2014 XL Latin American* , pp.1-12, 2014

[3] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi, and D. Hughes, “Using constraint programming to manage configurations in self-adaptive systems," IEEE Computer, vol. 45, no. 10, pp. 56-63, 2012.

[4] P. Grace, D. Hughes, B. Porter, G. S. Blair, G. Coulson, and F. Taiani, “Experiences with open overlays: a middleware approach to network heterogeneity,” Procs. 3rd ACM SIGOPS/European Conf. on Comput. Syst. 2008, 2008.

[5] N. Bencomo, P. Grace, C. Flores-Cortes, D. Hughes, and G. S. Blair, “Genie: supporting the model driven development of reflective, component-based adaptive systems,” in Procs. 30th Intl. Conf. on Softw. Eng., ACM, 2008, pp. 811-814.

[6] D. Hughes, P. Greenwood, G. Coulson, and G. Blair, “GridStix : Supporting Flood Prediction using Embedded Hardware and Next Generation Grid Middleware,” in World of Wireless, Intl. Symp. on Mobile and Multimedia Networks, 2006, pp.6 pp.626

[7] J. O. Kephart and D. M. Chess, “The Vision of Autonomic Computing,” *IEEE Computer Society*, no., pp. 41–50, 2003.

[8] R. Mazo, J. C. Muñoz-Fernández, L. Rincón, C. Salinesi, G. Tamura. VariaMos: an extensible tool for engineering (dynamic) product lines. 19th International Softw. Product Line Conf. (SPLC), Nashville-USA, 2015.

[9] R. Mazo, C. Salinesi, and D. Diaz. VariaMos: a Tool for Product Line Driven Systems Engineering with a Constraint Based Approach. 24th Intl. Conf. on Advanced Information Systems Engineering, pp. 1-8, 2012.

[10] P. Heymans, “The Albert II Specification Animator,” Technical Report CREWS 97-13, Cooperative Requirements Engineering with Scenarios, 1997

[11] M. Hussein, J. Han, A. Colman, and J. Yu, “An architecture-based approach to developing context-aware adaptive systems,” *Proc. - 2012 IEEE 19th Int. Conf. Work. Eng. Comput. Syst. ECBS 2012*, pp. 154–163, 2012.

[12] B. Morin, O. Barais, and J. Jézéquel. Models@ run.time to support dynamic adaptation. *Computer*, pp. 46–53. 2009

[13] N. Esfahani, A. Elkhodary, and S. Malek, “A Learning-Based Framework for Engineering Feature-Oriented Self-Adaptive Software Systems,” *IEEE Trans. Softw. Eng.*, vol. 39, no. 11, pp. 1467–1493, 2013.

[14] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas, “DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems,” in *Software Engineering for Self-Adaptive Systems II*, vol. 7475 LNCS, pp. 265–293, 2013.