A Stability-Aware Approach to Continuous Self-adaptation of Data-Intensive Systems

Marco Mori[1](✉), Anthony Cleve[1], and Paola Inverardi[2]

[1] PReCISe Research Center, University of Namur, Namur, Belgium
{marco.mori,anthony.cleve}@unamur.be
[2] Dipartimento di Informatica, University of L'Aquila, L'Aquila, Italy
paola.inverardi@di.univaq.it

Abstract. Nowadays data-intensive software systems have to meet user expectations in ever-changing execution environments. The increasing space of possible context states and the limited capacity of mobile devices make no longer possible to incorporate all necessary software functionalities and data in the system. Instead, the system database has to be adapted to successive context changes, in order to include all the information required at each stage. This adaptation process may translate into frequent and costly reconfigurations, in turn affecting negatively system stability and performance. This paper presents an approach to context-dependent database reconfiguration that aims to improve system stability by anticipating future information needs. The latter are specified by means of an annotated probabilistic task model, where each state is associated with a database subset. Experiments suggest that this approach has a positive impact on the stability of the system, the gain depending on the degree of similarity of the successive tasks in terms of database usage.

1 Introduction

In the era of ubiquitous environments, modern *data-intensive systems* are highly dynamic with respect to different aspects: they have to provide different software functionalities according to changing environmental conditions and changing user needs (i.e., context). Consequently, they have to provide users with the information that are suited for their current context of operation in a resource-constraint environment. The literature of *self-adaptive systems* [3,7,17] promotes the creation of self-adaptable applications by means of different software alternatives that satisfy different sets of requirements based on the current context. Different software alternatives, created for different contexts, possibly need different portions of data belonging to a big data source. Keeping all these data on the server and making them available through the *cloud* is not a good solution for variants running in resource-constraint environments. Indeed, continuous interactions with the server require an always-on Internet connection, which in turn consumes device resources. Further, in case of data that are rarely changed or whose modifications do not negatively affect competing accesses, frequent interactions with the server are even not necessary. For this reasons, we consider a local copy of the database available to the application at a specific context. Ubiquitous resource-constraint environments where these applications run, suffer from the limited capacity (storage and computational) of devices and from the large number of context situations for which different portions of data are necessary. This implies that it is not feasible to include the global database within the storage-constraint device thus making it necessary to provide adaptivity not only for applications but also for data and their manipulation mechanisms. In addition, even privacy concerns should prevent users, under certain conditions, to access sensitive information. To this end, the literature of *context-aware databases* includes the support to provide the application with the subset of the database according to current context, user tasks and user preferences [1,5,9].

Variability of accessing data poses the interest towards new data-intensive systems for ubiquitous environments that are able to perform run-time reconfigurations to data-related artifacts [12]. Data adaptations occurring in non-stationary environments suffer from performance degradations and furthermore they negatively affect *stability* of data, i.e., the capacity of the system to have the lowest possible variation of data with respect to the variations of tasks [8,14]. In this paper, we provide a framework for improving the stability of the database reconfiguration process by exploiting a predictive task model which expresses probable future variation of the application requirements in terms of variation of the information needs. Upon task variation we determine which are the *admissible* configurations of the database, i.e., the ones that fit into the device and that provide at least the minimum set of data required for the current task. Among those *legal* configurations, we then choose the one that allows reaching the highest level of stability. To this end, we consider two criteria: (i) how good each solution is with respect to future variations to the current task (and required data) and (ii) how good each solution is with respect to the operations to perform over the current database instance. It is worth noticing that a good configuration for the future may be too costly to reach from the current one. Conversely, a configuration that is easy to obtain may not constitute a good choice for the future. In light of this, we have formalized the two conflicting criteria and we have combined them in a single utility function with the aim of choosing the best admissible one by means of a future-aware and a future un-aware decision-making approach. Experimental results evaluate the stability for the two approaches with different input task models, each providing a different level of shared information needs and different memory occupancy among states.

Motivating Scenario. We consider an e-health system to support the doctor activities within and outside the hospital. A doctor works at the cardiology and orthopedy department either as physician or as director. Based on the current context, i.e., current location, role and activity, the doctor is interested in different excerpt of data. As physician performing check-up visits, the doctor needs to access a complete set of patient information with sensitive data, while if he performs check-up visits at the patient home (outside the hospital) he visualizes only basics patient information without sensitive data. Indeed, for security reasons, the application prevents the access to sensitive data in case of remote accesses. Only in case of operations at the surgery room, the doctor has to visualize information supporting the specific operation he is performing. As director, the doctor needs to access information about other doctors such as actual working time and operations performed during the week. In addition he may be interested in checking performance at the hospital level such as the average time of surgery for patients and the current stock of healthcare materials. Finally, in case of an emergency the doctor should only access basic patient information concerning particular diseases affecting the patient.

All the heterogeneous data that are necessary to perform all the doctor activities are included in a global data source. In a certain context, the doctor only requires a partition of the global data to perform its current activity. These activities are performed following a certain schedule, e.g., the doctor issues orders to healthcare manufacturer after he has discovered that a particular healthcare material is ending or he performs surgery operations after he performed check-up visits. Determining a deterministic scheduling for these activities is not possible since they depend on un-predictable conditions such as patients' behavior and materials usage. Nevertheless predicting how context changes (along with required data), may support the reconfiguration process in anticipating reconfiguration needs and assuring a better stability to data provided to the device.

2 Basics Models

Our approach supports the reconfiguration of the current subset of data based on the current and probable future contexts. We support the context-dependent variability of the conceptual schema following a feature engineering perspective. We model *features* as the basic functionalities of the system each corresponding to a certain *requirement*, a *contextual presence condition*, and a *data excerpt* expressed in terms of a set of entity types belonging to the global *conceptual schema*. We exploit a *probabilistic task model* where each state represents a different task for which a set of features with the corresponding data are required. Based on the current task we determine the *admissible* subsets of data and we apply a multi-objective optimization problem in order to choose the best possible reconfiguration which optimizes the overall system stability.

2.1 Data, Context and Features

Conceptual Schema. We represent *data* through a conceptual schema CS defined as a set of entity types and relationships. We model the variability of data by means of views, i.e., $V \subseteq CS$. Our granularity of adaptation is currently limited to entity types and relationships.

Context. We define the context as a set of dimensions affecting the interest of the user towards different portions of data, i.e., user role, task, location, device characteristics, etc. each of which can assume a finite set of domain values. In our e-health scenario we defined three context dimensions, i.e., user role ($d_1 = \{Doctor, Director\}$), user location ($d_2 = \{Cardiology, Orthopedy, Outside\}$) and activity ($d_3 = \{Surgery, Check{-}up, HospitalManagement, IssuingOrders, StockManagement, Emergency\}$).

Feature. We represent the context-dependent variability of data based on features, each defined as a triple $f = (Q, P, E)$ where Q is a functional, non-functional or a specific quality requirement (context independent), e.g., $f_x.Q$: *The system manages check-up visits*; P is the presence condition, i.e., a contextual constraint requirement which expresses the applicability of the feature, e.g., $f_x.P$: *Activity=Check-up AND Role=Doctor*, meaning that f_x is required in case of Check-up visits of the doctor, while E is the subset of entity types of the conceptual schema of interest for the feature, e.g., $f_x.E : \{Patient, Disease, Therapy, Diagnosis\}$. Among the features for the e-health scenario, *Surgery Operation* supports the operations at the surgery room, $BasicInfo$ manages basic personal information of the patient, *Cardiology* and *Orthopedy* features provide the support for managing patients belonging to the corresponding department, $SensitiveInfo$ manages sensitive patient information such as HIV test results and genetic screening information, *Hospital Dashboard* and *Doctor Dashboard* features support the direction activity of the doctor by providing useful information about care processes.

Given a subset T over the complete set of features we derive its corresponding database view V_T by creating a consistent excerpt of the global conceptual schema. In [11] we have formalized this process which consists of collecting together the entity types belonging to each feature in T and in adding relationships types and is-a relationships in order to produce a consistent view of the conceptual schema. Finally, we define function $M(T)$ to assesses the space required by the database instance corresponding to view V_T.

2.2 Probabilistic Task Model

Given a set of features F, the probabilistic task model is defined as $PTM = (S, L, Z)$ where $S = \{S_0, ..., S_t\}$ is the set of states with the required features, i.e., $S_i \subseteq F$ for $i = 1, ..., t$; L is the set of transition probabilities from one state to another; $Z : S \times L \rightarrow S$ is the probabilistic transition function. This automaton can be obtained by mining the past data accesses of the system. Such a mining process is out of the scope of this paper, which assumes the availability of the probabilistic task model. We introduce the notion of uncertainty in switching from one state to another. Following the idea of the PageRank algorithm we consider a low factor d of moving from one state to another while we consider with weight $1 - d$ the transition probabilities as obtained by the mining process. This allows us to partially consider transitions that have never occurred in past

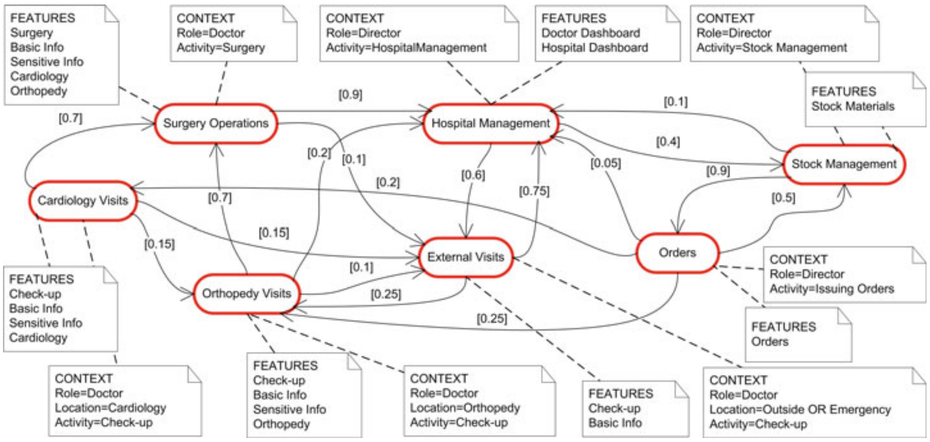

Fig. 1. Task model

executions and to have a fully connected automaton where infinite cycles are not possible (thus making it possible to evaluate a steady-state probability vector). Figure 1 shows the automaton for the e-health scenario. Among task states, *Hospital Management* manages care processes concerning a single doctor and the whole hospital thus it requires features *Doctor Dashboard* and *Hospital Dashboard*; *Surgery Operations* task supports the operations and it requires features *Surgery*, *Basic Info*, *Sensitive Info*, *Cardiology* and *Orthopedy*; state *External Visits* supports check-up visits of the doctor at the patients home and it requires features *Check-up* and *Basic Info*; *Cardiology Visits* and *Orthopedy Visits* states support check-up visits at the corresponding two hospital departments. Transition probabilities leaving from a state express the future tasks for the doctor/director, e.g., after *Cardiology Visits* he will either perform *Orthopedy Visits* or *External Visits* with equal probability or he will perform *Surgery Operations* with an higher probability.

Database Reconfiguration Cost. Reconfigurations to the current database instance come at some cost which affects the performance of the application. This cost depends on the operations that have to be performed over the current database and it can be approximated with the number of operations that have to be performed over the conceptual schema. To this end, we have implemented an algorithm that given two subsets of features $F_i, F_j \in 2^F$ and their corresponding views V_i, V_j, it evaluates $C(F_i, F_j)$ as the number of entity types and relationships to add and to delete to switch from V_i to V_j.

Stability. Our definition of stability takes into account observed reconfiguration costs along with the variations of data required at a certain context. Given a certain path of states of the probabilistic automaton $Path = \{S_0, S_1, ..., S_n\}$

each containing a set of required features $S_i \subseteq F$ for $i = 1, ..., n$, we define *stability* as the ratio between observed database reconfiguration costs (output) and required database reconfiguration costs (input) (Eq. 1). The component at the numerator sums the costs for reconfiguring the *best* configuration T_i at step i while the denominator sums costs between states of the path.

$$st = \frac{\sum_{i=0}^{n-1} C(T_i, T_{i+1})}{\sum_{i=0}^{n-1} C(S_i, S_{i+1})} \tag{1}$$

Sharing Index. The nature of the task model is essential for evaluating the best approach for improving stability during the reconfiguration process. To this end, given two subsets of features $F_i, F_j \in 2^F$ and their corresponding views V_i, V_j, we define the *data-requirement sharing* as the number of their shared elements $D_{Equal}(F_i, F_j) = |V_i \cap V_j|$. Starting from this metric, we define the sharing index sh as the average number of shared elements among states of the probabilistic automaton (Eq. 2). For each state we sum the data-requirement sharing D_{Equal} with respect to any of the possible next states weighted with the probability of moving towards any of those. Then we sum these quantities weighted with the importance of each state according to the steady-state probability vector r. Finally, we divide this quantity by the maximum number of elements per state $MaxES$.

$$sh = \frac{\sum_{i=1}^{t} r_i \cdot \sum_{j=1}^{t} p(i,j) \cdot D_{Equal}(S_i, S_j)}{MaxES} \tag{2}$$

Memory Occupancy Index. We also characterize the automaton according to the percentage of memory required at each state (Eq. 3) as the ratio between the total space required by the features of each state S_i (weighted with its importance according to vector r) and the space available at the device $MaxSpace$.

$$mo = \frac{\sum_{i=1}^{t} r_i \cdot M(S_i)}{MaxSpace} \tag{3}$$

Data-Requirement Distance. In our approach we consider how good a certain configuration is with respect to future states of the task model. To this end, given two set of features F_i, F_j and their views V_i, V_j, we define their *data-requirement distance* $D_{Miss}(F_i, F_j) = |V_j - V_i|$ as the number of elements that lack to V_j with respect to V_i.

3 Approach to Stability

Upon task variation we have to determine the best possible reconfiguration of data. In Eq. 4 we formalize this decision-making problem as a minimum optimization problem to assess the most suitable configuration of features among the ones that fit the memory space limit $MaxSpace$ while providing the set of features required at the current task S_{curr}. The utility function evaluates the best T that minimizes two conflicting objectives: its reconfiguration cost with respect to the current one T_{curr} and its distance with respect to probable future states.

$$\begin{array}{ll} \min\limits_{T \in 2^F} & \alpha \cdot C(T_{curr}, T) + (1-\alpha) \cdot D_{Future}(w, T) \\ \text{subject to} & M(T) \le MaxSpace, S_{curr} \subseteq T \end{array} \tag{4}$$

To weigh two conflicting objectives we introduce the parameter $\alpha = [0, 1]$. Setting values closer to 1 we give more importance to the current reconfiguration cost, while setting values closer to 0 we give more importance to future states. Based on the values assigned to α and w we compare different approaches to the decision-making problem. By setting $\alpha = 1$ we obtain a *future un-aware technique* where only the current state and the cost between source and target configuration are considered. On the contrary, by setting a value of α lower than 1, we obtain a *future-aware technique*. In this case we exploit w to consider till one step ahead, till two steps ahead, and so on. We evaluate the fitness of a configuration with respect to future states by considering a variable number of steps ahead in the future. If we consider one step ahead ($w = 1$) we evaluate the sum of the data-requirement distances between the target configuration and each next state weighted with the probability of moving from the current state to each next state:

$$D_{Future}(1, T) = \sum_{i=1}^{t} p(S_{curr}, S_i) \cdot D_{Miss}(T, S_i) \tag{5}$$

If we consider two steps ahead ($w = 2$) we augment the previous sum with the relevance for states that are two steps ahead:

$$D_{Future}(2, T) = D_{Future}(1, T) + \sum_{i=1}^{t} \sum_{j=1}^{t} p(S_{curr}, S_i) \cdot p(S_i, S_j) \cdot D_{Miss}(T, S_j) \tag{6}$$

We have also defined a technique to consider an increasing number of states by evaluating the fitness of each T as the sum of its distances to each state S_i weighted according to the steady-state probability vector r:

$$D_{Future}(fixpoint, T) = \sum_{i=1}^{t} r_i \cdot D_{Miss}(T, S_i) \tag{7}$$

Example. We suppose that the doctor is at state *Orthopedy Visits* (Fig. 1) since he is performing check-up visits at the Orthopedy department of the hospital. Upon task variation towards *Hospital Management* we have to reconfigure data to include features required in the new state, i.e., *Doctor Dashboard* and *Hospital Dashboard*. Since the space for data is limited, we have to discard almost all the features that are not anymore required at the new state in order to give room for the data required by the two new features. Among the admissible configurations that contain the new two features we choose the one that minimizes stability considering current and future context changes. Indeed, by looking at future tasks we discover that with high probability (0.6) feature *Check-up* will be required again by the task *External Visits*, thus we maintain it by choosing among the admissible configurations one that contains features *Doctor Dashboard*, *Hospital Dashboard* and *Check-up*. Thus, we improve stability of data by keeping a feature that is likely to be required by future tasks.

4 Validation

We have implemented a simulator of the theoretical framework with the aim of measuring the impact of future-aware and future un-aware approaches on the stability of the system. We consider as input to the simulator two different large conceptual schema: $OsCommerce$ conceptual schema which supports the management of an e-commerce web store[1], and $Oscar$ conceptual schema which supports the management of heterogeneous care processes within a hospital[2]. $OsCommerce$ conceptual schema, which contains 330 entity types and 813 relationships, is available at http://www.info.fundp.ac.be/~mmo/osCommerce.use through the UML-based tool USE [6]. $Oscar$ conceptual schema which contains 455 entity types and 267 relationships, is available at http://www.info.fundp.ac.be/~mmo/Oscar.lun through the data-modeling tool DB-MAIN[3]. Starting from the above schemas we have considered a set of different experiments with the following characteristics. We have generated 12 features by assigning to each of them 12 different entity types. Then we have created different automata by randomly selecting 4 features for each state and a distribution of random probabilities each one between 0.1 and 0.3. For each automaton we have experimented a set of 100 paths each one with 1000 hops representing variations to the current task state. For each path we have measured its stability and we have computed an average stability value among all paths. We have repeated the same experiment with different combinations of values for α and w to compare future-aware and future un-aware approaches.

We have obtained different stability values based on the nature of the input automaton, depending on its sharing index sh and its memory occupancy index mo. Figure 2 shows the stability measure obtained according to six different automata which are good representative over the set of experiments we carried out. These automata have an average of either 33 % ($sh = 0.33$) or 45 % ($sh = 0.45$) of shared elements between states and either 90 % ($mo = 0.90$) or 70 % ($mo = 0.70$) or 40 % ($mo = 0.40$) of memory required at each state. The stability curves refer to the *future un-aware* ($\alpha = 1$) approach and to future-aware approaches, i.e., $1-$*step ahead* ($w = 1$), $2-$*step ahead* ($w = 2$) and $fixpoint$. If the memory required at each single state is almost equal to the memory available ($mo = 0.9$), it follows that st is approximately equal to 1 for both future-aware and future un-aware approaches. In this case, each time a state requires

[1] OsCommerce technical guide, http://guifre.lsi.upc.edu/OSCommerce.pdf.

[2] Oscar official website, http://www.new.oscarmanual.org/.

[3] DB-MAIN official website, http://www.db-main.be.

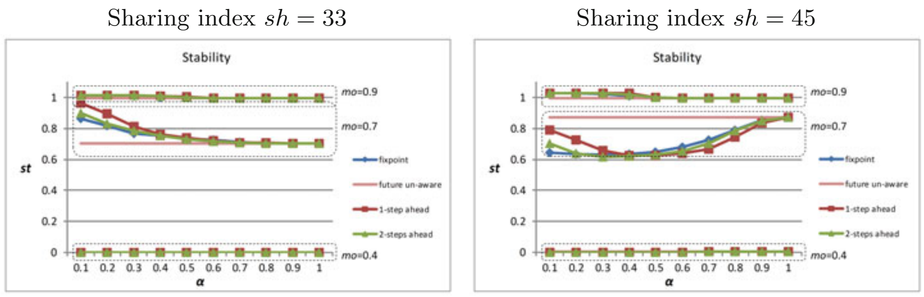

Fig. 2. *Stability* of future un-aware and future-aware ($w = 1$, $w = 2$, $w = fixpoint$) techniques depending on utility objective weights α with 33 % and 45 % of sharing and 90 %, 70 % and 40 % of memory occupancy.

database elements that were not required at the previous state it is necessary to perform the corresponding operations to change the schema. This behavior holds independently to the percentage of shared elements among states. If the memory occupancy among states is equal to 70 % ($mo = 0.7$), we obtain different results depending on the sharing index. First, with $sh = 33$ the best stability is obtained by applying the *future un-aware* technique; in particular, while after $\alpha \geq 0.6$ we have similar stability values, with $\alpha \leq 0.6$ future-aware approaches produce higher values for st, meaning that on average given the same paths of task variations, they required higher database reconfiguration costs. This occurs because the gain of loading features that will be required in future states is less than the gain obtained by choosing the less costly reconfiguration. Second, results gained with $sh = 45$ show that future-aware approaches produce better stability than future un-aware ones; in particular with $\alpha \geq 0.6$ already looking one-step ahead we get on average the lowest database reconfiguration costs given the same paths of task variations. Finally, the stability curves with 40 % of memory occupancy ($mo = 0.4$) show that there is space for keeping a big set of features which is enough to satisfy almost each future task. Even though future un-aware approaches will load this set of features gradually (as needed), while future-aware approaches load them in advance (before they are needed), both approaches produce a stability close to 0 meaning that on average almost no reconfigurations to the schema are performed.

As we observed in the three different cases of memory occupancy, we claim that if the required memory at each different state is too near ($mo = 0.9$) or too far ($mo = 0.4$) from the total capacity, it is not convenient to apply a future-aware approach because the same level of stability can be already obtained with a simple future un-aware approach. In both cases, the level of sharing elements of the automata does not affect the level of stability. Conversely, if the memory required at each state is far from both the maximum and the minimum capacity (e.g., $mo = 0.7$) it may be convenient to adopt a future-aware approach. In particular, we observed that adopting future-aware techniques is convenient if states share a high number of elements, as it can be evaluated with our sh

index. Furthermore, these approaches are particularly suitable in case of unstable contexts with frequent and intensive variations from one task to another.

5 Related Work

Task based models are receiving an increasing attention in supporting the adaptivity of systems with the aim of providing better performance to the reconfiguration process [18]. Task models are not fixed during the whole lifecycle process but they may change as a consequence of context variations [16] or user preferences variations [10, 19]. Different approaches provide adaptivity to data based on context or user preferences variations. In [2] the authors propose a methodology to extract and to merge portions of the relational and logical schema based on a hierarchical context model. In [5] the authors propose an approach to select the most suitable portion of the relational database based on context-dependent user preferences. In [20] the authors define a filtering technique for extracting the excerpt of the conceptual schema based on a set of entity types that are required in a certain context. All these approaches to database adaptivity [2, 5, 20] provide the subset of the database based on a certain input context. Nevertheless continuous reconfigurations of the database are not supported, i.e., a certain excerpt of the database is produced based on context and kept for all the system lifetime. Applications for ubiquitous environment need continuous self-adaptations while taking into account device capacity limits and multi-objective criteria (costs and performance) in choosing the best possible reconfiguration [15]. Moreover, in order to achieve reconfigurations of data resilient to changes it is important to consider future context variations. This problem has been addressed in the literature of self-adaptive systems, e.g., in [4] predictive availability of context resources are exploited to achieve system adaptations, while in [13] adaptations are achieved based on a probabilistic user preferences model. Differently from these approaches, we provide adaptivity to data and we achieve reconfigurations of the current database that are resilient to future probable changes of information requirements as annotated in a predictive task model.

6 Conclusion and Future Work

We presented a framework to support continuous reconfigurations of data-intensive systems in resource-constraint environments. We formalized a decision-making problem to select the best possible reconfiguration of data according to a predictive task model expressing future probable variations of data accesses. Results showed that it makes sense to consider a predictive task model for improving stability of the reconfiguration process and they showed under which conditions of the task model it is convenient to adopt either a future-aware or a future un-aware approach. As for future work, we will implement the reconfiguration process by adopting the DB-MAIN tool for creating sub-schemas of the global schema and a relational DBMS for propagating these variations to actual data. We will adopt higher level of granularity for adapting the schema by considering variations of attributes beyond concepts and relationships. Furthermore, we will introduce further variability dimensions to the reconfiguration process such as the addition of un-anticipated features with data, the addition of new states and the variation to the probabilities of the task model.

Acknowledgment. This work has been partially supported by first author's FSR Incoming Post-doctoral Fellowship of the *Académie universitaire 'Louvain'*, co-funded by the Marie Curie Actions of the European Commission.

References

1. Bolchini, C., Curino, C., Orsi, G., Quintarelli, E., Rossato, R., Schreiber, F.A., Tanca, L.: And what can context do for data? ACM **52**(11), 136–140 (2009)
2. Bolchini, C., Quintarelli, E., Tanca, L.: Carve: context-aware automatic view definition over relational databases. IS **38**(1), 45–67 (2012)
3. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.): SEFSAS 2009. LNCS, vol. 5525. Springer, Heidelberg (2009)
4. Cheng, S.-W., Poladian, V.V., Garlan, D., Schmerl, B.: Improving architecture-based self-adaptation through resource prediction. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) SEFSAS 2009. LNCS, vol. 5525, pp. 71–88. Springer, Heidelberg (2009)
5. Ciaccia, P., Torlone, R.: Modeling the propagation of user preferences. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 304–317. Springer, Heidelberg (2011)
6. Gogolla, M., Büttner, F., Richters, M.: Use: a UML-based specification environment for validating UML and OCL. SCP **69**(1–3), 27–34 (2007)
7. Inverardi, P., Mori, M.: A software lifecycle process to support consistent evolutions. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Self-Adaptive Systems. LNCS, vol. 7475, pp. 239–264. Springer, Heidelberg (2013)
8. Karsai, G., Lédeczi, A., Sztipanovits, J., Péceli, G., Simon, G., Kovácsházy, T.: An approach to self-adaptive software based on supervisory control. In: Laddaga, R., Shrobe, H.E., Robertson, P. (eds.) IWSAS 2001. LNCS, vol. 2614, pp. 24–38. Springer, Heidelberg (2003)
9. Martinenghi, D., Torlone, R.: A logical approach to context-aware databases. In: D'Atri, A., De Marco, M., Braccini, A.M., Cabiddu, F. (eds.) Management of the Interconnected World, pp. 211–219. Physica-Verlag HD, Heidelberg (2010)
10. Miele, A., Quintarelli, E., Tanca, L.: A methodology for preference-based personalization of contextual data. In: EDBT, pp. 287–298 (2009)
11. Mori, M., Cleve, A.: Feature-based adaptation of database schemas. In: Machado, R.J., Maciel, R.S.P., Rubin, J., Botterweck, G. (eds.) MOMPES 2012. LNCS, vol. 7706, pp. 85–105. Springer, Heidelberg (2013)
12. Mori, M., Cleve, A.: Towards highly adaptive data-intensive systems: a research agenda. In: Franch, X., Soffer, P. (eds.) CAiSE 2013 Workshops. LNBIP, vol. 148, pp. 386–401. Springer, Heidelberg (2013)
13. Mori, M., Li, F., Dorn, C., Inverardi, P., Dustdar, S.: Leveraging state-based user preferences in context-aware reconfigurations for self-adaptive systems. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 286–301. Springer, Heidelberg (2011)

14. Nzekwa, R., Rouvoy, R., Seinturier, L.: A flexible context stabilization approach for self-adaptive application. In: PerCom, pp. 7–12 (2010)
15. Parra, C., Romero, D., Mosser, S., Rouvoy, R., Duchien, L., Seinturier, L.: Using constraint-based optimization and variability to support continuous self-adaptation. In: SAC, pp. 486–491 (2012)
16. Quintarelli, E., Rabosio, E., Tanca, L.: Context schema evolution in context-aware data management. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 290–303. Springer, Heidelberg (2011)
17. Salehie, M., Tahvildari, L.: Self-adaptive software: landscape and research challenges. TAAS **4**(2), 1–42 (2009)
18. Sousa, J.P., Poladian, V., Garlan, D., Schmerl, B., Shaw, M.: Task-based adaptation for ubiquitous computing. Trans. Sys. Man Cyber (C) **36**(3), 328–340 (2006)
19. Sykes, D., Heaven, W., Magee, J., Kramer, J.: Exploiting non-functional preferences in architectural adaptation for self-managed systems. In: SAC, pp. 431–438 (2010)
20. Villegas, A., Olivé, A.: A method for filtering large conceptual schemas. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 247–260. Springer, Heidelberg (2010)