Divide and Conquer – Scalability and Variability for Adaptive Middleware

Ulrich Scholz
European Media Lab GmbH
Schloss-Wolfsbrunnenweg 33
69118 Heidelberg, Germany
ulrich.scholz@eml-d.villa-bosch.de

Romain Rouvoy
University of Oslo
P.O. Box 1080 Blindern
0316 Oslo, Norway
rouvoy@ifi.uio.no

ABSTRACT

In this paper, we introduce a divide and conquer approach for the adaptation of distributed applications management by a potentially large number of interacting middleware instances. The method aims at a quick partitioning of the overall problem into smaller sub-problems that can be solved almost independently. The partitioning is found by symbolic reasoning and by applying expert knowledge that is encoded in explicit rules.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design – *distributed systems*

General Terms

Algorithms, Performance, Design

Keywords

Adaptive middleware, hierarchical decomposition, symbolic reasoning

1. INTRODUCTION

In this work we consider the problem of adapting large-scale distributed applications in a mobile environment with multiple users and frequent context changes, to maintain a high utility to the user. We consider such problems within MUSIC [1], an initiative to develop a comprehensive open-source middleware that facilitates the development of self-adaptive software.

MUSIC aims at a large-scale deployment of multiple middleware instances in parallel. Some of these instances interact, form clusters, and run (parts of) applications. Applications that run on such clusters are adapted together. Each such middleware cluster, or each set of related applications, is seen as an adaptation *scenario*. However, there is no one-to-one mapping between applications and middleware instances, *i.e.*, applications can be spread out over several clusters and a cluster can host several applications. Furthermore, the topology of middleware clusters is transient and can change, merge, and split at any time. For such a collection of scenarios, *i.e.*, large transient clusters of partially dependent, partially independent middleware instances, applications, devices, connections, and other artefacts related to adaptation, we assign the term *theatre*.

Current solutions to the adaptation problem use a global approach: They consider all alternative configurations of all applications at once and choose the one that yields the best utility. For large theatres, this approach is not feasible due to the combinatorial explosion of the number of alternative configurations. As a consequence, we have to go beyond a global adaptation approach.

Thus, we propose a *Divide and Conquer* (D&C) approach that provides both adaptation scalability and variability. The main goal is the immediate and quick breakdown of an overall theatre into smaller, logically and physically independent sub-problems that can be handled separately and in parallel. D&C works if the adaptation problems of the parts are small and independent enough to be solved, and if the combined solution to these parts is good enough to satisfy the users. In other words, D&C does not aim at finding a global optimal solution to the adaptation problem but at a scaling and variable one that is considered as good enough (or feasible).

In the remainder of the paper we first present existing approaches to the adaptation problem and then describe the formal definition of this problem. We continue with introducing the divide and conquer approach and explain its realization. Before the concluding remarks we give some examples of its application.

2. RELATED WORK

Existing techniques to the adaptation of variation points usually use global approaches.

Brute force algorithms systematically explore the adaptation space of a scenario and evaluate all the alternative configurations [2]. Such algorithms terminate when all the configurations have been evaluated. They are then able to provide the optimal configuration that solves the initial problem but do not support scalability in terms of configuration alternatives, because the number of alternative configurations directly influences the computation time.

Dominating factors methods are concerned about improving the performance of adaptation heuristics [3]. These methods aim at reducing the adaptation space based on a ranking of alternative configurations. The ranking is determined by an evaluation of the dominating factors of each configuration. A *dominating factor* represents a constant utility over context variations. The ranking acts as precompiled knowledge of configurations that are likely to solve the problem in an optimal manner. This knowledge allows discarding a part of the alternative configurations if their dominating factors are below the current best utility found by the heuristic. Thus, the efficiency of this method relies on the accuracy of the dominating factors identified.

Arshad *et. al.* [4] present a work that uses an AI planning approach in pervasive computing to model tasks and a policy engine that generates the plan. While this work demonstrates the feasibility of the approach, its scalability is still an open issue.

3. ADAPTATION OF LARGE THEATRES

The formal definition of adaptation allows us to reason about its properties: optimality, scalability, variability, and locality.

3.1 The Variation Point Problem

We define adaptation in the following way: A *variation point* is a part of an application that can be realized in different ways, *e.g.*, with different pieces of code; we say it can be assigned different *variants*. *Resolving* all variation points, *i.e.*, assigning variants to all variation points of an application yields a *configuration* of that application. Although many configurations deliver the same required functionality to the user (*e.g.*, the capability of video streaming) they often differ in non-functional properties (*e.g.*, power consumption). The perceived match of these non-functional properties to the desires of the user is the *utility* of a configuration (provided that the functional requirements are met).

One goal of the middleware is to maintain the behaviour of the application such that the user requirements are satisfied: If the *context* of the application, *i.e.*, its environment, its resources, and the user goals, changes then the application should be adapted in such a way that the new configuration yields the best utility of all feasible configurations. This task is called the *variation point problem*.

Not all of the assignments to the variation points of an application yield a valid configuration because variation points can depend on each other: For example, an existential dependency between two variation points holds if resolving one variable only makes sense if a particular choice was made for the other, such as a variant that introduces variation points itself. Another example is an architectural dependency where two variants (of two different variation points) are mutually exclusive or require each other.

3.2 Optimality vs. Scalability

Optimality and scalability are two properties of systems: The former is its quality for a single user; the latter is its ability to provide a basic service to an increasing number of users.

From the viewpoint of a user, the adaptation goal of a system is independent of the size of the variation point problem: The user wants an optimal adaptation of its applications to the context. As discussed in Section 2, optimal solutions are found by considering all variation points of a theatre and by finding and assigning the solution that yields the best utility. Independently of the size of a particular theatre these approaches to find optimal adaptations remain the same.

However, for large theatres, finding an overall optimal solution is not feasible. One reason is the combinatorial explosion of alternative configurations based on the number of variation points: If every variation point has 5 variants then an application with 5 variation points has 3125 alternatives while an application with 10 variation points already has about 10 million. Although increasing processing power, restrictions on valid choices of variants, and heuristics allow solving large variation point problems, even medium sized theatres are often infeasible for a global method.

Because an optimal solution does not scale, the best we can hope for is a solution to the variation point problem that serves many users quickly in a way that satisfies each user.

3.3 Variability and Locality

Among other reasons that prevent the use of global solutions for the variation point problem is the irregular and transient topology of theatres, the unreliability of connections, and the interest of users to stay autonomous. In the case of global solutions, a small change in one part of a theatre causes the whole theatre to be adapted. An adaptation approach with local variability can keep the changes local instead.

If, for example, all applications are local to a device (and the device has enough resources) then there is no need to use an external adaptation mechanism: Local adaptation means a tolerance to connection failures, a reduced response time, an increased privacy, and cheaper connection cost. In short: Users like local decisions to be decided locally.

Nevertheless, if the resources of the device become insufficient, and if applications use multiple devices and serve multiple users then a combined adaptation of several applications and devices is necessary. Therefore, the adaptation mechanism has to form clusters of middleware instances and has to single out an adaptation master for each cluster.

When the context changes, *i.e.*, in case of failing connections, changes in the device topology, and changing user preferences, the decisions regarding clusters and their masters have to be changed, too: Middleware clusters are merged and split, and different masters need to be assigned, which means that the adaptation mechanism itself has to adapt. With a D&C approach, the changes caused by local context events can remain local to some degree, and areas of the theatre with unchanged context can remain unaffected.

4. DIVIDE AND CONQUER

We now propose the D&C approach as a scalable, variable, and local technique for the adaptation of large theatres. Below, a *solver* denotes an entity that finds an acceptable solution of adaptation problems. The overall principles of its design are as follows: (1) Do independent tasks in parallel; (2) assign the solver and the device that matches the problem (power of device, problem complexity, required quality, ...); (3) decide on local variation points locally; (4) adjust to changes of the topology, *i.e.*, have the ability to revoke decisions; and (5) only change decisions when it is necessary.

All but the fourth point address scalability, while the last three deal with variability. Furthermore, the third point increases fault tolerance by reducing the dependency on network connections.

4.1 The Divide and Conquer Approach

As we have seen, an overall optimal solution to the variation point problem is not always feasible. And for large theatres even a suboptimal overall solution is unlikely. Therefore, we propose a two-step approach: First decompose the overall problem into small, mostly independent subparts. Then solve the subparts, *i.e.*, assign variants to its variation points.

We want to distinguish the mechanisms involved in the steps of D&C: The second step is done by *solvers*. D&C can use a range of available solvers (brute force, etc.) and we will not elaborate on them in this work; some are described in the related works section. The first step is achieved by *controllers*. They decide about the decomposition and assign solvers, but they do not solve variation points, *i.e.*, assign variants.

Decomposition should be fast and yield good utility, which means that it cannot be based on pure reasoning alone: It has to use knowledge that is provided *a priori*. Instead of using a specialized method with built-in knowledge, we propose to use a symbolic AI reasoner and explicit knowledge. The latter is given to the system via strategies, which are described in Section 5.2

4.2 Building the Decomposition Tree

Decomposition is done by forming sets of variation points and partitioning them into subsets. The *decomposition tree* represents complete information about all such decision made at all nodes for a particular theatre. Each node of the decomposition tree is (annotated with) a variation point set, its root is the initial set that comprises all available variation points, and the children of a node represent the decomposition set of their parent. (For simplicity we identify the variation point set of a node with its node.) Decomposition is hierarchical and proceeds recursively until leaf nodes are reached.

All nodes of the decomposition tree have an associated controller. When triggered, it decides upon how to deal with the variation point set of its node. Among its options are choosing a solver and applying it on the set (only for leaf nodes), further decomposing the set, or triggering the adaptation of its parent node (*i.e.*, triggering the controller of its parent node).

There are two kinds of leaf nodes, namely *leaf partition nodes* and *negotiation nodes*. By *inner nodes* we refer to nodes that are not leaf nodes. Leaf partition nodes partition the set of variation points, *i.e.*, every variation point of the overall problem is assigned to exactly one leaf partition node. The idea of the D&C method is that the variation point sets of leaf partition nodes are solvable almost independently of other variation points.

Negotiation nodes encapsulate all necessary interactions between partitions, for example when partitions compete for bandwidth and memory resources. They contain variation points of two or more leaf partition nodes. Variation points outside a particular negotiation node should not (directly) depend on the resource associated with this node, *i.e.*, the negotiation group should only cause their reassignment if it cannot be solved.

For the decomposition tree, adaptation means to reconsider earlier decisions, starting at the leaf nodes and moving upwards towards the root when satisfactory solutions cannot be found at lower node levels and depending on controller logic.

4.3 Assigning Controllers to Devices

The device hosting the middleware instance that controls a node is called the *master* of that node. (Usually, exactly one middleware instance runs on each device. For simplicity we use these terms interchangeably.) Most of the time, the master is assigned by the controller of the parent node. In case the parent node is not available (*e.g.*, at system start or after a network outage) the available middleware instances are found by a discovery mechanism. The hosting instance is then assigned by a simple negotiation (*e.g.*, voting).

If a node contains variation points of applications that are only deployed on a single device then this device is a candidate to be the node's master. If a node contains variation points of several devices then the powerful ones among them are candidates. If all devices of the cluster are too weak then the master has to be a different server "close" to the node (connection wise).

Assigning controllers to different middleware instances causes the decomposition tree to be distributed. Controllers only have knowledge about their own decisions but not, e.g., about how its children partition their assigned variation point sets further. Therefore, no single middleware instance holds the complete information about a decomposition tree at a certain time.

4.4 Scalability and Variability

The main goal of the D&C approach is to reach scalability and variability. D&C is scalable under the assumption that theatres are composed of almost independent sets of variation points whose size is nearly constant compared to the size of the theatre. This assumption seems to be justified where users employ private devices and applications across a large number of devices are rare (see the Radio Ballet example in Section 6.1). Here, D&C allows adaptation to remain local, independent of the overall number of variation points.

D&C is variable because it can adapt to changes in the topology. If connections fail and a cluster of devices become isolated then their variation points form a decomposition tree of their own. If instead two decomposition trees have to join because a new application spans variation points of both then it is likely that only those leave partition nodes have to be adapted that are involved or related to that adaptation.

5. REALIZING DIVIDE AND CONQUER

The D&C approach to the variation point problem is an application of symbolic reasoning, *i.e.*, an approach where we have a symbolic representation of the system state and explicit knowledge on how to manipulate the system. To provide this functionality, the D&C strategy requires the following: (1) A model of the variation point problem that allows its decomposition and enables us to reason about this decomposition. (2) Decomposition strategies, *i.e.*, *a priori* knowledge on how to decompose the variation point set in given theatres. (3) A machinery to support and administer the decomposition of variation point sets, their solution, their reunion, and the negotiating between them.

5.1 Modelling Hierarchical Decomposition

The D&C approach is based on a distributed symbolic model of the problem, the context, and the intermediate steps and decisions of arriving at a solution. The model allows expressing the state of the problem and its solution at a certain time, *i.e.*, what is true and what is false at that time.

Relevant aspects are encoded as predicates. For example, the possible assignment of variants to variation points could be encoded by the predicate *assignable* and the fact that a variant v_1

can be assigned to a variation point vp_3 is expressed by the ground predicate *assignable*(v_1, vp_3) being true. The constants v_1 and vp_3 are symbols referring to the actual variant and variation point in the problem, hence the name *symbolic* model.

The model can also use real numbers and inequalities to model (*e.g.*, resources and utilities). For example, the predicate *available_mem*(d_3, *mem*) expresses the available memory of device d_3 and the inequality $mem \geq req\_mem_1 + req\_mem_2$ restricts the combined required memory usage of applications 1 and 2 ($used\_mem_1$ and $used\_mem_2$, respectively) accordingly.

One part of the model describes the problem, *i.e.*, the variation points and their variants, to which application a variation point belongs to, constraints on the assignments of variants, resources and their usage by applications, and utilities. It also allows expressing the context aspects relevant to D&C: The utility of applications to the user, the device topology and the available connections, and the current time, among others.

The model also describes the internal state of the D&C method, such as the assignments of variants to variation points, the assignments of variation points to a node, the distribution of nodes onto devices, the assignment of controllers and solvers to nodes, the assignment of nodes to middleware instances (devices), and the parent and the children of a node.

5.2 The Decomposition Strategies

Decomposition strategies aim at establishing the decomposition tree of a theatre by defining how variation point sets are partitioned and which controllers are assigned to nodes. A strategy consists of a collection of rules that are triggered by the occurrence of a scenario in a theatre and describes how this scenario can be resolved and exploited by D&C. In other words, the strategies determine when and how theatres can be decomposed.

It is important to note that strategies are defined independent of specific scenarios; they are more like a screenplay and define the general ways in which decomposition takes place. Instead of saying that a specific device should be adapted by an algorithm X the strategy only says that it should be controlled by an algorithm with specific properties. This makes it possible to use only few decomposition strategies to support a large set of possible decompositions which can all occur and change at runtime.

Strategies are explicit knowledge, *i.e.*, stated in a format usable by a general problem solving method. Therefore, they are easily changeable, which allows to adjust D&C without changing the underlying reasoners and the underlying partitioning machinery. Different strategies allow the same configuration of the middleware to exhibit different behaviours, *e.g.*, to apply different solvers or to distribute variation points over devices in a different way. The same flexibility is not available if the strategies are implemented implicitly as part of an algorithm. Here, changing any part of the strategies would require exchanging some part of the application containing the algorithm, so small adjustments would result in large changes.

Strategies depend on insights into the workings of the middleware, the adaptation, the devices, and the applications. They are created by middleware experts and possibly by the designers of the applications running on the middleware, based on their knowledge of theatres, scenarios, and users.

One source of strategies is the dependency between variants and between variation points: If an existential dependency holds between two variation points (*e.g.*, architectural constraints) then they are likely in the same leaf partition node. If two variants of two leaf partition nodes are mutual exclusive then they are in a common negotiation node. Strategies will likely separate different devices, applications, and users. In other words, if two variation points belong to the same application, they are likely to be in the same leaf partition node. The same is true for two variation points of two applications that are deployed onto the same device, of the same application, and of two applications used by the same user.

5.3 The Underlying Mechanics

The middleware has to provide several components and services for the D&C approach to be realized: An *encoder* transforms the current situation of the system into a world model and a D&C problem. If the system changes, it changes the world model and the problem accordingly. The controller of a node applies a *reasoner* to its local problem. The reasoner solves the decomposition problem by matching strategies against the current world model. The solution of a local D&C problem has to be transformed back into actions of the middleware: For example, applying a solver to a variation point set, handing over parts of a problem to other middleware instances, and combining several parts of a problem to a larger one. Further services include the detection and management of context changes (*e.g.*, device discovery), and the communication between middleware instances.

We see several options to realize a reasoner, for example rule based systems and constraint programming. Currently, we favour its realization by *Hierarchical Task Network* (HTN) planning [5], a form of AI planning. In short, AI planning is concerned with changing a world from its current state into a state with desired properties by applying a sequence of actions, *i.e.*, transformations that change the world. HTN planners are AI planning systems that are based on tasks and their decomposition into subtasks, and hence are suitable for the divide and conquer approach.

6. EXAMPLE SCENARIOS

We exemplify divide and conquer by three scenarios.

6.1 Radio Ballet

Imagine a music festival with many users using mobile devices, *e.g.*, smart phones, computing infrastructure provided by the organisers, and companies with promotion stalls. One company provides a fascinating new entertainment: radio ballet.

Radio ballet is a group activity in which visitors can participate by running a "radio" application on their device. Sometimes, when enough participants are near the company's stall, the servers initiate a radio ballet: The devices hand over the control to one of the companies' servers, which adapt them such that they can be used as channel to instruct the user to participate in a ballet-like choreography. Participating is a lot of fun and demonstrative, which gives the stall much visibility. At the end of the ballet, the servers release the control over the mobile devices, not after handing out tokens for free gifts as gratitude and incentive.

The adaptation mechanism also has the option to not participate: For example, in the case of low battery power, the middleware might decide in favour of extended operating time.

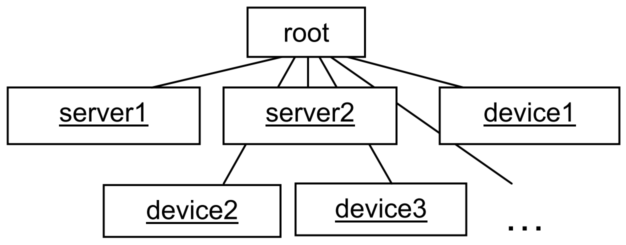

Figure 1: Decomposition tree before the adaptation.

Figure 1 shows the decomposition tree before the adaptation. The theatre comprises many mobile devices (three are shown) and two servers. On each device, a leaf partition node holds all variation points on that device and the middleware instance on that device is the master of that node (depicted by underlining). After the adaptation, depicted in Figure 2, the servers control leaf partition nodes that comprise several mobile devices. The middleware of device 2 does not take part in the radio ballet, so it has its own leaf partition node.

How does D&C find this hierarchy? Initially, the decisions were made locally and in parallel, according to the first and the third principle. After the context change, *i.e.*, after the servers initiated the radio ballet, the best solution is to associate the participating mobile devices with the most powerful servers and to hand over the control (principles two, four, and five).

6.2 Further Example Scenarios

In the second scenario, a single user works with a single mobile device and all the applications are local to this device. Network connection is available but unused; the adaptation between the applications on the mobile device is done locally. After a context change, one of the applications needs more computing power, which exceeds the capabilities of the device.

According to the third principle the decisions were made locally before the context change: The variation points of the applications running on the local device form one leaf partition node. After the context change, the local controller decides "cannot adapt" and hands over the control to its parent (principles four and five). The controller of the parent node creates a new partitioning of its variation points and the new leaf partition node includes the variation points of the mobile device and those of some servers. The solver of this new leaf partition node decides to migrate code from the mobile device to the server (principle two).

In the third scenario, two users share an application. Their devices are connected via WLAN and UMTS and the entire application is adapted as a whole by a middleware instance on one of the devices. Now, the common application is terminated and the users are suddenly independent of each other: They do not share any application, device, and resource.

Principles one and three suggest to separate both users and to adapt the applications of each user by the middleware instance running on the device local to its respective user. Before the adaptation, each user has its own leaf partition node. The network and the common application are adapted via negotiation nodes. After terminating the common application, the control is handed over to the parent node, which recreates the same leaf partitions (including local controller and solver) but this time without negotiation nodes.

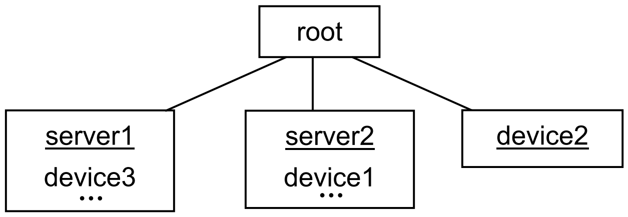

Figure 2: Decomposition tree after the adaptation.

7. CONCLUSIONS

In this paper, we propose the application of the Divide and Conquer (D&C) approach to solve the adaptation of large-scale systems (theatres) over context changes. This approach aims at distributing a given variant point problem by decomposing the theatre into independent subparts. Dedicated solvers, which act on problems of a smaller size, then adapt these parts separately. The composition of adapted parts is controlled by this approach to resolve possible conflicts. To implement this D&C approach, we propose to use symbolic reasoning to encode the decomposition knowledge as modular rules that can be easily extended. These rules select also the best solver for each subpart of the theatre.

The work on D&C is in a preliminary stage and many problems are still open. For example, the current design assumes that the overall utility of the adaptation is maximized by a fair adaptation of the variation points of each user. We plan to investigate how theatres that include players having selfish goals can be handled. Another open question is how reasonable is the assumption that theatres consist of clusters of variation points whose size is independent of the size of the theatre. Also the design of the symbolic model and the strategies is not finalized.

We plan our next step to be a first model of a theatre together with the corresponding strategies to demonstrate the viability of the approach.

8. ACKNOWLEDGMENTS

The work is funded by the Klaus Tschira Foundation and by the European Commission under the contract number 035166 for the MUSIC project. We would like to thank Frank Eliassen, Svein Hallsteinsen, Gunnar Brataas, Eli Gjørven, Yun Ding, and the ESSPE'07 reviewers for their helpful comments.

9. REFERENCES

[1] IST MUSIC project. www.ist-music.eu

[2] IST MADAM project. "*Theory of Adaptation*". Deliverable 2.2. December 2006. p. 44–49. www.ist-madam.org

[3] Sousa, J.P. *et. al.* "*Task-Based Adaptation for Ubiquitous Computing*". IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 36(3), 2006.

[4] Arshad, N., Heimbigner, D., Wolf, A.L.: *Deployment and dynamic reconfiguration planning for distributed software systems*. In: proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, USA, IEEE Computer Society (2003) 39.

[5] Ghallab, M., Nau, D., and Traverso, D., "*Automated Planning – Theory and Praxis". Morgan Kaufmann*. San Francisco, CA, 2004.