HAL
open science

SAFDIS: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications

Guillaume Gauvrit, Erwan Daubert, Françoise André

To cite this version:

Guillaume Gauvrit, Erwan Daubert, Françoise André. SAFDIS: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications. 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Sep 2010, Lille, France. pp.211–218, 10.1109/SEAA.2010.25 . inria-00480221

HAL Id: inria-00480221

https://inria.hal.science/inria-00480221v1

Submitted on 28 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications

SAFDIS: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications

Guillaume Gauvrit, Erwan Daubert, Françoise André
IRISA / INRIA
Campus de Beaulieu, 35042 Rennes cedex, France
{Guillaume.Gauvrit, Erwan.Daubert, Francoise.Andre}@irisa.fr

***Abstract*—Service Based Applications (SBA) running in distributed and heterogeneous environments are subject to varying constraints that can lead to fluctuations in the quality of the application. We propose a solution in the form of a distributed framework for adaptation to improve in a autonomous way the quality delivered by those applications and to maintain it above a minimum level. This framework, named SAFDIS for Self-Adaptation For DIstributed Services, enables the dynamic evolution of service-based architectures by providing all the functionalities of the MAPE model. Among these functionalities, particular emphasis is put on the analysis phase which permits to use several reasoners able to take decisions with multiple temporal scopes, at short term as well as at long term. Specific attention is also paid to the planning phase, which enables to schedule parallel actions while taking into account different constraints.**

***Keywords*-Self-adaptation; Distributed Applications; SOA; Distributed Adaptation Framework;**

I. Introduction

Applications are more and more build as a composition of services running on large scale, dynamic and heterogeneous environments. Indeed, Service-Oriented Architectures enforce a strong separation of concerns through the loose coupling of the services. Moreover, the use of communication protocols specific to long distance communications provides a programming paradigm well suited to make distributed applications. However, those distributed applications can be subject to variations in the quality of service they provide. This dynamism is inherent to the nature of these applications and of their running environment. It can be due to various factors, such as the volatility of execution nodes (that arise from failures, maintenance actions or voluntary connections or disconnections), the evolution of the services composing the application (new services may be deployed, existing ones removed), varying user load and varying users demands that require different levels of quality of service. All these factors lead to the necessity for dynamic (i.e. at run time) adaptation, without human intervention, at the level of one service, one application, the infrastructure or the environment.

The existing solutions to build service-oriented applications, such as the various OSGi and SCA implementations, offer basic tools that can be used to develop adaptation, but without providing specific means for self-adaptation of applications.

In order to provide a solution to this need of an adaptation support we propose a distributed context-aware framework, which follows a generic design, to support the dynamic evolution of distributed service-oriented applications. This framework, named SAFDIS for Self-Adaptation For DIstributed Services, allows to analyse with various depth the events impacting the application execution, then to schedule and execute distributed adaptation actions.

This article is organised as follow. Section II gives an overview of the SAFDIS framework. An example follows in Section III to illustrate an adaptation using SAFDIS. The process of an adaptation is then detailed step by step in Section IV. Next, some relevant details of our implementation of SAFDIS are given in Section V. After that some related works are discussed in Section VI. Finally, Section VII concludes this article and presents our future works.

II. Framework

Today's society depends on software system used in the everyday life such as bank or airport systems. These systems have to be available 24/7 and have to take care of the environmental changes or users requirements. They are more and more built using Service-Based Applications (SBA). Dynamic adaptation is used to adapt these kind of systems without needing to interrupt their execution. This concept has grown in several research areas such as mobile computing and grid computing. But introducing facilities for adaptation in executing code is a very difficult task and designing new applications taking into account the situations where adaptation may occur is almost impossible.

To tackle this problem, we chose to make a framework built separately from the functional code, dedicated to run-time adaptation, as described by the *MAPE* model [1] and capable to evolve itself. We propose this solution for Service-Based Applications.

Our framework, called SAFDIS (Self-Adaptation For DIstributed Services) is divided into the four main functionalities of the *MAPE* model. *Monitoring* is the observation function to detect changes that imply adaptation. When a change is detected, the monitoring phase triggers the

978-0-7695-4170-9/10 $26.00 © 2010 IEEE
DOI 10.1109/SEAA.2010.25

IEEE computer society

analysis to analyse it and find an adaptation strategy if it is required. Then this strategy is given to the *planning* phase to compute a schedule of actions that will satisfy the strategy. The last step is the *execution* of the schedule to reconfigure the system (application, services and the environment). Between the four phases and the services supervised by the framework, an adaptation manager is used to make the link between the different parts of the adaptation system and the services to adapt.

Since our framework is built to adapt service-based applications, we chose to also build it with services. Each function is provided by a different service in order to benefit from the advantages of loose coupling, such as updating one of them at run time without modifying the others. Our framework can be distributed upon all execution nodes where applications that our framework have to adapt are located. Thus, the framework is divided in multiple autonomous and cooperating instances. It is also fully decentralized, meaning there are no instances with privileges or special purposes. This design avoids single points of failure and makes the framework scalable. This also allows to distribute some operations that can be computationally heavy such as analysis.

In order to address the necessities of adaptation at the level of the service-based application, the infrastructure and the environment, SAFDIS can monitor and execute adaptation actions on the services, the service-oriented platform and the operating system. However this paper focus on the application level.

The different phases of SAFDIS are described more precisely in Section IV, taking the migration of service to illustrate how an adaptation action is performed. This illustrative example upon which we experiment the use of our framework is described in the following section.

III. Illustrative Example

To illustrate the use of the SAFDIS framework, this section presents a simple example of an adaptation in a distributed environment.

Let us consider a video conference service used in a multi-site corporation. This corporation have a pool of servers to run business specific applications or services. An instance of SAFDIS is deployed on every server of this pool.

The video conference service is hosted on one of these servers. Its main functionality is to collect the video stream of the participants of a video conference and send back a video stream showing a mosaic of the other participants. A similar process is done with the audio stream but is not detailed here for the sake of simplicity. Creating a mosaic involves to re-encode the stream. The service supports to encode the mosaic in different formats, such as MPEG 2 and MPEG 4, the format being chosen by the service requester. The main difference between the formats are that MPEG 2 takes more bandwidth while MPEG 4 takes more computing power to encode.

In this use case, initially a few persons are in a video conference from their respective offices using the MPEG 2 encoding format. Later, another person joins the conference using a smartphone while traveling. Since the smartphone bandwidth is limited and since the hardware supports the decoding of the MPEG 4 format, this format is selected. This means that the video conference service has to encode the mosaic stream in MPEG 2 and MPEG 4, increasing its needs in computing power. However the server hosting the service is not powerful enough, so the encoding algorithms start to drop frames from the video stream.

The SAFDIS instance running on the same server as the video conference service notices that the CPU consumption is at its maximum and that the service is dropping frames. From this information, the SAFDIS instance decides to migrate the service on another server. A negotiation starts with the other SAFDIS instances to know if some of the servers can host the video conference service. Among the answers, a server with more computing power available is chosen. After that, the original SAFDIS instance plans the actions needed to migrate the video conference service. The migration involves actions distributed and synchronized between the old and the new server. After the migration the video conference service is restarted. No more frames are droped.

The various services used in the example, as well as SAFDIS, are implemented using the OSGi [2] service-oriented platform. One platform is runing on each servers, each of them hosting a SAFDIS instance.

This illustrative example is detailed when it is relevant in the following sections of the paper. Implementation details are given in section V.

IV. Adaptation Chain

This section presents step by step the process of an adaptation using SAFDIS. The illustrative example is used at the end of each these steps.

A. Information Gathering

The monitoring function is used to provide an informative and dynamic view of the adaptive entity and its environment to the other functions of SAFDIS. Thus, it is the starting point of every adaptation undertaken.

SAFDIS pictures low-level local views of the system, picking relevant information from the service-oriented platforms, the adaptive services, the operating system and the hardware. Since SAFDIS is distributed, there is one local view per instance of SAFDIS. Those views are pictured by values representing states of the context. Complex values are computed to make synthetic high-level pictures. SAFDIS can probe both passively or actively the system to generate events and update the view. The pieces of information that have to be gathered are specified by the other functions of the framework.

To generate events, SAFDIS uses multiple ad hoc *probes*, which are the software or hardware elements taking the measures. The procfs virtual file system under Unix-like OSs (usually mounted at `/proc`) is an example of an OS level probe. Other probes can be used at the level of the services and of the service-oriented platforms. Those probes are listened to and queried by *monitors* which provide a unified service interface to the probes. Every monitor can listen and query multiple probes and have to be able to answer to requests on the values it monitor. Each instance of SAFDIS has an *event manager* that uses the monitors to gather the events, compose them and keep a local view of the system. The event manager can make computations on an event over a window of time, such as the maximum time to process a request over the last minute. Those computations are described in a dedicated language. When changes occur in the view they are notified to the analysis function of SAFDIS. The analysis and planning functions can request values of events to the event manager, to get complementary information when needed.

In our video conference example, the events generated on the platform hosting the transcoding service are notifications of skipped frames by the transcoding service, the number of frames skipped over the last second and the last ten seconds, the ratio of CPU usage over the last second and the last ten seconds. Skipped frames inform of a lack of processing power to convert video streams while CPU usage informs of the tolerance of CPU load peaks. How these events are used depends on the policy of the analysis phase.

B. Distributed Analysis

The analysis function of a MAPE adaptation system has two goals. The first goal is to identify situations needing an adaptation. It listens to updates of the view (events) of the system pictured by the event manager. Then it analyses the changes in the system and decides if an adaptation is needed consecutively to this change. The second goal of the analysis function is to make an adaptation decision when a need arises.

However, in SAFDIS, in order to achieve these goals the analysis function has two objectives. The first one is to be able to take decisions with multiple temporal scopes. This is the ability to either react fast or to take proactive decisions for the long term. This implies the ability to analyse the context with a variable depth of reasoning. The second objective is to distribute and decentralise the analysis process. This enables to spread the computational load and make the analysis process scalable.

Figure 1 illustrates the architecture of the analysis function in SAFDIS. This function is provided as a service by the analyst composite. Inside this composite, the decision maker component answers to the two goals of the analysis by analysing the context and computing the strategies. Our objective to take decisions with multiple temporal scopes is answered by the use of different reasoners used concurrently by the decision maker. Having reasoners with different algorithms enables to analyse the context with different depth. The deeper the reasoning, the more proactive it can be but also the less it becomes able to react fast to a new situation. Our second objective, distributing and decentralizing the analysis, is answered by the distribution of instances of SAFDIS and through the coordination of the analyst composites. This coordination is done by the negotiation manager and the negotiator components. The negotiator component handles the negotiation protocol while the negotiation manager coordinates the negotiations involving multiple peers.

Instead of trying to picture a global view of every element contributing to the application, which would consume communication resources and not be scalable, SAFDIS pictures multiple local views. This means that the instances of the analysis component have to take decisions based on partial knowledge of the system. This knowledge alone is not always enough to make adaptation strategies. Thus, the decision function uses negotiation mechanisms in order for them to cooperate in the decision taking process.

The decision maker listens to events coming from the event manager and sends them to the reasoners for analyse. If they produce a strategy, it means that there is a need to adapt. Since the goals of the possible adaptations depend on the elements to adapt, an adaptation policy has to be written by the services or applications administrators. Indeed, some may seek energy efficiency while others the best quality of service possible for the end-user and this cannot be guessed.

As shown by Figure 1, in our implementation we use two reasoners, one makes short-term decisions and the other handles the long-term ones. The short-term reasoner makes fast and simple decisions and is useful when rapid reactions are needed. It uses a simple Event-Condition-Action (ECA) algorithm. The other is slower to analyse situations but can make complex strategies and optimize the application for the long-term. It is done with a generic algorithm based on utility functions [3]. More details are given in Section V and in [4].

When a decision service notices a change in its view of the system that necessitates an adaptation, the decision service issues a strategy stating the changes to make. If those changes do not involve peers, the strategy is handled directly to the planning function. In the other case the decision maker handles the strategy to the negotiation manager to negotiate it.

The negotiation manager is the component managing the multiple negotiations that can happen at the same time. When a strategy involves multiple peers, the negotiation manager splits it into strategies involving only one peer, transmits them to the negotiator component, and collects the multiple replies into a single amended strategy. The negotiator component tracks strategies being negotiated be-

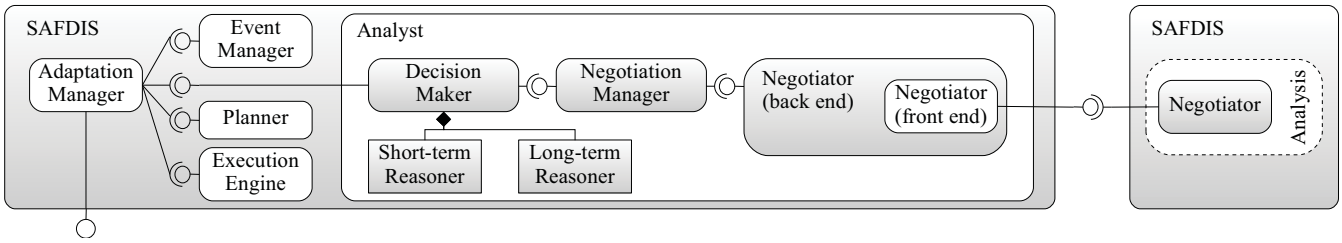

Figure 1. Analysis Component in the framework

tween its SAFDIS instance and other instances. It enforces that the negotiation protocol is followed. At the end of negotiation, the initial distributed strategy may have been accepted, rejected, modified or extended.

When a negotiator receives a strategy initiated by a peer, it uses the reasoning capabilities of its decision maker to negotiate it. This decision maker can accept to apply this strategy, make a counter proposal or reject it. In order to apply such a strategy, the decision maker might need the involvement of other services. So, it can initiate another strategy and wait for the answers of those involved before making its decision on the original strategy. Therefore the process of making a strategy is recursive. To avoid that the negotiation takes too much time, the initiator of the strategy can set an expiration date to the negotiation. The potential cycles in this process are avoided by using an unique identifier by strategy. Once a distributed strategy is approved by every peers, the initiator can choose to apply it or dismiss it in whole or in part. Being able to dismiss part of a strategy allows to make evaluations of proposals and choose one of them.

The length of a negotiation can be shortened when the partners developing the services agree on a set of available strategies. This negotiation process is even optional. Indeed, SAFDIS can be used to adapt a service independently of its related services (providers or users) and thus it can be used to build an autonomous service. In such a configuration, the negotiation manager and negotiator components can be disabled. Similarly, the reasoners can work for one service without knowledge of the surrounding services. However in this case a strategy impacting other services cannot be made, as for example migrating the provider of another service.

In our example of a video conference application, the SAFDIS service managing the video conference service notices it lacks computational resources to perform its tasks. Therefore it issues a strategy stating to migrate the service to each potential execution node. When every peers have replied, it chooses the best offer among those who accepted their part of the distributed strategy or made a counter proposal. Then it dismisses the parts of the strategy related to the other peers, thus modifying the strategy to involve only the selected peer. The strategy is then accepted and applied by the new host.

C. Strategy Planning

As we explain in the previous section, the analysis phase produces a strategy. A strategy specifies a new state to reach for the service, the application or the environment (i.e. the system). The strategy doesn't precise how to make the adaptation but what the goal of the adaptation action is. More precisely, it is the role of the planning phase to find a set of actions allowing to reconfigure the system.

Until now the planning phase has received little attention in the context of adaptation and in many cases the planning algorithms used produce simple orderings of actions. In these cases, the result is not very efficient since the execution can take more time than necessary because the actions are totally ordered. Moreover in distributed environments, where actions can be asynchronous, it is necessary to add some synchronisation actions to ensure the predefined order.

These simple algorithms do not usually take into account other specificities of the system. Indeed, the processor load, the amount of memory or the bandwidth used during the execution of the adaptation can be of importance. For example, even though it is possible to concurrently move two services from one execution node to another, the overload of the network between these nodes may be incompatible with the quality of service to be preserved for the other activities.

Some works have been done in research topics such as artificial intelligence or control theory to build planning algorithms that allow to compute schedules of actions. These general purpose planning algorithms can take into account constraints like the duration of the execution or the resource consumption. Some of those algorithms are presented in Section VI. Since all algorithms do not provide the same facilities, we do not choose a specific algorithm for our planning engine but we offer a way to automatically select an efficient algorithm relatively to the needs.

The planning engine, illustrated by Figure 2, is composed by a manager using services that provide different planning algorithms. The manager selects one of the available algorithms according to constraints defined by user needs or by administrator requirements. Once an algorithm is chosen, the manager converts the strategy into a *problem* to solve by the planning algorithm. The strategy represents the source and target configurations which are respectively the configuration of the system before the adaptation and the configuration the

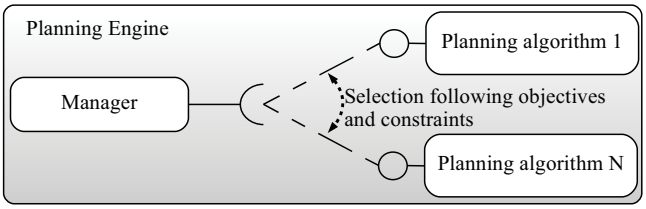

Figure 2. Planning Engine

system have to reach through the adaptation. The problem is another representation of these configurations specified for the planning engine. It contains an initial (or current) state which corresponds to the source configuration and a goal state which corresponds to the target configuration. These two states are used as the input to the planning algorithm service which then returns a schedule of actions.

We have implemented a translation scheme that can automatically translate the description of the source (and target) configurations into the initial (respectively goal) states of the planning algorithms, by using appropriate translations. More details on this translation are given in Section V.

Most of the time the actions used for adaptation are statically defined, but in a Service-Oriented Architecture the actions might be provided by services and thus they can disappear during the life-cycle of the application. The execution engine notifies the planning engine when an action appears or disappears.

To link the planning and execution phases, two different representations of the actions have been made. Indeed, the information on an action needed by the planning phase are not the same than those needed for the execution.

So for the planning phase we use an abstract representation of the concrete actions. These abstractions define the information needed to plan the strategy, that are pre- and post-conditions, parameters and execution properties of the actions.

On the one hand, with these abstract actions several properties of the actions are hidden. For example, the planning engine does not need to know which Service-Oriented Architecture is used to implement the actions, as shown by Figure 3. On the other hand, some information are added to help the planning engine to compute the plans. For example, if the planning algorithm needs to know the duration of an action to compute the most efficient plan, this information can be given by the execution engine.

In our illustrative example, the strategy specifies the new configuration to reach by the system. From the descriptions of the current state of the system and the state to reach – differing in the location of the video conference service – the planning engine finds one action to migrate the service between two nodes but also a set of structural actions which modify bindings between the streaming server and the clients. First, the video conference service is duplicated

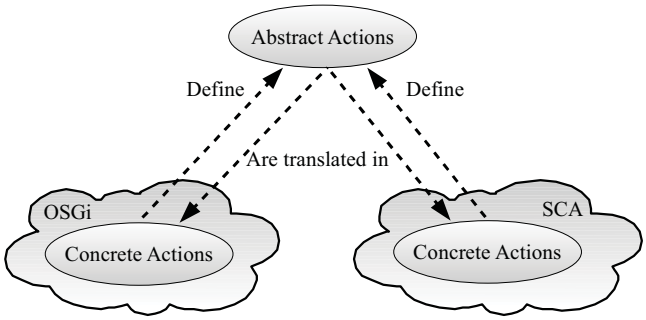

Figure 3. Abstract and concrete actions

to the new node. Then, bindings between this service and the others are changed to replace the old service with the new one. Finally the old service is stopped.

D. Execution of Plans & Migration of Services

Once the planning engine has computed the action plan from the strategy, the execution engine is called to adapt the service, the application or the environment.

From the schedule, the execution engine looks for concrete actions matching the abstract actions. As written previously in IV-C, various concrete actions match the same abstract action. Therefore the execution engine needs to choose the best one according to constraints (e.g.: duration, resource consumption...).

Unlike the other phases, the execution phase cannot be generic, since it is in direct relation with the application implementation. So in this part we choose to illustrate the execution by the migration action, as it is typical of the execution of a service based application in a distributed environment. Moreover, it is the action needed in our illustrative example. This action is done at the level of the service-oriented platform.

The migration of service consists in moving a service from an execution node to another. The migration is useful for example when the first node (called node A thereafter) is overloaded by too many services. If one service, in our example the transcoding video stream service, is migrated to another node (node B), resources previously used by this service on the node A are released and can be reallocated for the others. Migration can also be useful to improve the quality of the migrated service if it is migrated on a node that is not overloaded.

The abstract actions are dependent from the service-oriented architectures have to be implemented into concrete actions for each of the supported architectures.

In order to present the migration used in our example, this paragraph presents succinctly the OSGi platform [2], which is a standard service-based architecture. Each application built on top of this platform can discover and use services provided by applications located on the same platform.

Services on other platforms can be discovered using an external registry. Service-oriented applications using OSGi are divided on several bundles. A bundle is a library component which packages services that are logically related. A bundle imports and exports Java packages and offers or requires services.

To migrate a service, one has to migrate the bundle that contains it. OSGi platforms enable to install a bundle using an URL of the JAR file representing the bundle.

Once the bundle is migrated, it is started and it can register all its services. But the adaptation strategy might state that only one service have to be registered on the new platform. Our implementation provides this option by enabling the bundle to ask which services require registration. In addition, this implementation requires to be able to save and reload the current state of the service. This is done using the *memento* design pattern.

Finally, when the service is registered on the new platform (on node B), it is unregistered on the previous platform (on node A), then the bundle is stopped if all its services have been unregistered.

V. Implementation Details

This section presents some details and technical choices of our implementation.

SAFDIS is a service-oriented framework. Our implementation is built for the OSGi platform, using iPOJO [5] to manage the life-cycle and the binding of the SAFDIS components. OSGi provides some adaptation actions, such as dynamically registering and unregistering the services, however they are not sufficient for the needs of self-organising service-oriented applications.

The event manager of the monitoring phase uses the *WildCAT* [6] framework to compose events. It provides the means to compute the complex values needed to build a high-level representation of the context.

The short-term reasoner uses an ad hoc Event-condition-Action algorithm. However, it is designed to use also a rule-based engine compliant with the Java Rule Engine API (JSR-94). So users of SAFDIS would be able to reuse their knowledge of the API to write analysis policies. Among the possible implementations of the API, JESS [7] or JRuleEngine [8] can be used for instance.

The algorithm used by the long-term reasoner is based on utility functions [3]. They are functions computing the *utility* of a configuration of a system, enabling to compare different configurations. The algorithm explores the space of possible configurations and provides a strategy when it finds one with an utility significantly higher than the utility of the current configuration. We have developed a similar algorithm for the dynamic adaptation of master-worker patterns [9].

The various SAFDIS instances communicates through the service provided by the negotiator components. This service is implemented as a web service using the SOAP protocol so that it can be accessible from other platforms. In order to expose the negotiation service as a web service while keeping the benefits of iPOJO to manage the life-cycle of the negotiator component, this component is split in two parts, as shown on Figure 1. The back end is managed by iPOJO and contains the business logic. It creates the front-end component exposing the web service. The negotiation follows the Iterated Contract Net Protocol [10], issued from works on agent systems.

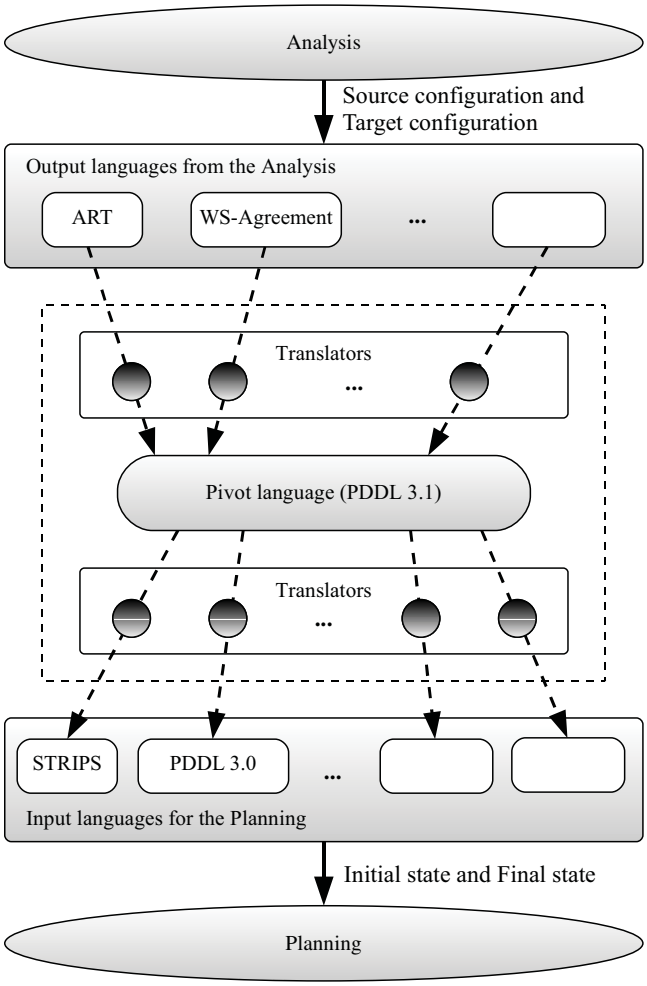

Figure 4. Translation between strategies and planning input

In the planning phase, for the use case described here, the Graphplan [11] algorithm is used. This algorithm is able to exhibit parallel actions according to precondition actions. However we experimented other algorithms, such as SGplan [12] and Prodigy [13].

Since the languages in which strategies can be written are different from the languages used by the various planning algorithms, a translation scheme is established between the languages. The PDDL [14] language is used as a pivot language as illustrated by the figure 4.

In the implementation of our illustrative example, the video conference service and the clients use the VLC software for the decoding and encoding of the video stream. The selection of the encoding format and the address of the stream are done trough the service interface. However, we plan to use the SIP protocol to initiate the video connection and negotiate the encoding format.

VI. Related Works

Currently, the research made to build frameworks for dynamic adaptation of software mostly targets component-based applications. Thus several frameworks have been designed for component architectures, such as Dynaco [15], SAFRAN [16] and Jade [17]. Only the first two of them are generic, meaning that they let free the choice of the adaptation logic. SAFRAN requires to use Wildcat for the monitoring and FScript to execute the strategy. Jade only makes use of the adaptation for fault tolerance purposes. The generic framework proposed in [18] has the particularity to be distributed and to adapt distributed but homogeneous component-based applications. All of those frameworks target the Fractal [19] component model.

Since those frameworks target components, they are not faced with the dynamism of service-oriented architectures. In particular the dynamic connection of the services allows to modify at run-time the structure of an application, a task that would be difficult to do with components.

For our implementation of SAFDIS, we chose the OSGi platform. However other platforms could be used. Among the platforms implementing the SCA specification, FraSCAti [20] provides basic support to build self-managing services. Like OSGi with iPOJO, FraSCAti can manage the components life-cycle. It also provides support for non-functional aspects by using metadata elements attached to components that can trigger non-functional services. Up to now, we are not aware of works based on FraSCAti that implement an adaptation framework.

The research done in the analysis domain is usually targeting specific contexts, either restricted to an application domain or to a given infrastructure. This enables them to provide efficient algorithms, such as [21], [22], [23], [24], but they are bound to particular scenarios.

There are nevertheless some more generic works: in [25] the authors propose a reasoning engine using aspect-oriented programming (AOP) to manage the variability of the system. Aspects are woven at run-time into a model reflecting the runtime system. This model enables to validate to some extent the adaptation to be done before impacting the system.

In the context of dynamic adaptation, the planning phase has received little attention. Pegasus [26], [27] is used to deploy components on computational grids. It use the PRODIGY [28] system to schedule the actions during deployments. In the same context, Sekitei [29] is a planning algorithm based on AI planning researches. But, to our knowledge, no adaptation system in service-oriented architectures uses general purpose planning algorithms.

VII. Conclusion and Future Works

Due to the nature of their use, distributed service-based applications are subject to varying levels of quality of service. Self-adaptation capabilities enable these applications to remedy to this situation. However, even though several research works have tackled the problem of software adaptation, few of them address this problem in distributed environment or in service-oriented architectures.

In this paper, we propose the SAFDIS framework to bring self-adaptation capabilities to distributed service-based applications. In particular, SAFDIS is able to make decisions with different temporal scopes, to make this analysis in a distributed way, to plan on the fly the adaptation actions and to target heterogeneous service platforms.

In order to improve SAFDIS, we plan to study the use of learning algorithms for the long-term reasoner and we wish to study more profoundly the coherence between the different reasoners in that case. In addition, to make the use of SAFDIS as seamless as possible, we intend to offer the ability to weave the code connecting SAFDIS into the adaptive services by using aspect-oriented programming.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] "OSGi Alliance Specifications," 2010. [Online]. Available: http://www.osgi.org/Specifications

[3] J. O. Kephart and R. Das, "Achieving self-management via utility functions," *IEEE Internet Computing*, vol. 11, pp. 40–48, 2007.

[4] F. André, E. Daubert, and G. Gauvrit, "Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures," in *5th International Conference on Internet and Web Applications and Services*, Barcelona, Spain, 2010, pp. 309–314.

[5] C. Escoffier, R. Hall, and P. Lalanda, "iPOJO: an extensible service-oriented component framework," in *IEEE International Conference on Services Computing (SCC)*, Salt Lake City, USA, 2007, pp. 474–481.

[6] P.-C. David and T. Ledoux, "WildCAT: a generic framework for context-aware applications," in *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing (MPAC)*, Grenoble, France, 2005, pp. 1–7.

[7] E. Friedman-Hill, *Jess in Action: Java Rule-Based Systems*, ser. In Action. Manning Publications Co., 2003.

[8] "Jruleengine project," 2010. [Online]. Available: http://jruleengine.sourceforge.net/

[9] F. André, G. Gauvrit, and C. Pérez, "Dynamic adaptation of the master-worker paradigm," in *Proceedings of the 9th IEEE International Conference on Computer and Information Technology*, vol. 2, Xiamen, China, 2009, pp. 185–190.

[10] "Fipa interacation protocol specifications," 2010. [Online]. Available: http://www.fipa.org/repository/ips.php3

[11] A. L. Blum and M. L. Furst, "Fast planning through planning graph analysis," *Artificial Intelligence*, vol. 90, pp. 281–300, 1995.

[12] Y. Chen, C. wei Hsu, and B. W. Wah, "Sgplan: Subgoal partitioning and resolution in planning," in *In Edelkamp*, 2004, pp. 30–32.

[13] E. Fink and M. Veloso, "Prodigy planning algorithm," Dept. of Computer Science, Carnegie Mellon University, Tech. Rep., 1994.

[14] M. Ghallab, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, and D. Weld, "PDDL – the planning domain definition language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[15] J. Buisson, F. André, and J.-L. Pazat, "Supporting adaptable applications in grid resource management systems," in *Proceedings of 8th IEEE/ACM International Conference on Grid Computing*, Austin, USA, 2007, pp. 58–65.

[16] P. C. David and T. Ledoux, "An aspect-oriented approach for developing self-adaptive fractal components," in *Software Composition*, ser. LNCS, vol. 4089, 2006, pp. 82–97.

[17] S. Bouchenak, F. Boyer, S. Krakowiak, D. Hagimont, A. Mos, S. Jean-Bernard, N. de Palma, and V. Quema, "Architecture-based autonomous repair management: An application to j2ee clusters," in *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems (SRDS)*, Orlando, Florida, 2005, pp. 13–24.

[18] F. André, M.-T. Segarra, and M. Zouari, "Distributed dynamic self-adaptation of data management in telemedicine applications," in *Proceedings of the 7th International Conference on Smart Homes and Health Telematics (ICOST)*, ser. LNCS, vol. 5597, 2009, pp. 303–306.

[19] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "The fractal component model and its support in java: Experiences with auto-adaptive and reconfigurable systems," *Software—Practice & Experience*, vol. 36, no. 11-12, pp. 1257–1284, 2006.

[20] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform," in *6th IEEE International Conference on Service Computing (SCC)*, Bangalore, India, 2009, pp. 268–275.

[21] G. Bastide, A. Seriai, and M. Oussalah, "A self-adaptation of software component structures in ubiquitous environments," in *Proceedings of the 5th ACM international conference on Pervasive services (ICPS)*, Sorrento, Italy, 2008, pp. 173–176.

[22] M. Al-Turkistany, A. Helal, and M. Schmalz, "Adaptive wireless thin-client model for mobile computing," *Wireless Communications & Mobile Computing*, vol. 9, no. 1, pp. 47–59, 2009.

[23] S. Vadhiyar and J. Dongarra, "Self adaptability in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 235–257, 2005.

[24] S. Wu and Y.-T. Chang, "A user-centered approach to active replica management in mobile environments," *IEEE Transactions on Mobile Computing*, vol. 5, no. 11, pp. 1606–1619, Nov. 2006.

[25] B. Morin, O. Barais, G. Nain, and J.-M. Jezequel, "Taming dynamically adaptive systems using models and aspects," in *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, Vancouver, Canada, 2009, pp. 122–132.

[26] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, S. Koranda, A. Lazzarini, G. Mehta, M. Papa, and K. Vahi, "Pegasus and the pulsar search: From metadata to execution on the grid," in *International conference on Parallel processing and applied mathematics (PPAM)*, ser. LNCS, vol. 3019, Czestochowa, Poland, 2004, pp. 821–830.

[27] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[28] J. Carbonell, O. Etzioni, Y. Gil, R. Joseph, C. Knoblock, S. Minton, and M. Veloso, "PRODIGY: an integrated architecture for planning and learning," *SIGART Bull.*, vol. 2, no. 4, pp. 51–55, 1991.

[29] T. Kichkaylo, A. Ivan, and V. Karamcheti, "Constrained component deployment in wide-area networks using AI planning techniques," in *Proceedings of the 17th IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2003.