DYNAMICALLY RECONFIGURABLE DATA-INTENSIVE SERVICE COMPOSITION

Onyeka Ezenwoye, Salome Busi
Electrical Engineering and Computer Science Department, South Dakota State University, Brookings, South Dakota, U.S.A.

S. Masoud Sadjadi
School of Computing and Information Sciences, Florida International University, Miami, Florida, U.S.A.

Keywords: Service composition models, Scientific workflow, Adpaptability, Dynamic reconfiguration, Choreography, Orchestration.

Abstract: The distributed nature of services poses significant challenges to building robust service-based applications. A major aspect of this challenge is finding a model of service integration that promotes ease of dynamic reconfiguration, in response to internal and external stimuli. Centralized models of composition are not conducive for data-intensive applications such as those in the scientific domain. Decentralized compositions are more complicated to manage especially since no service has a global view of the interaction. In this paper we identify the requirements for dynamic reconfiguration of data-intensive composite services. A hybrid composition model that combines the attributes of centralization and decentralization is proposed. We argue that this model promotes dynamic reconfiguration of data-intensive service compositions.

1 INTRODUCTION

A system is a set of interrelated components that are interacting in an interdependent manner to accomplish a common goal. A service can be defined as the function performed or the behavior exhibited by a system. Service Oriented Architecture (Bichler and Lin, 2006) provides service based computing which deals with the interactions between loosely coupled, autonomous and replaceable components that are available over a heterogeneous network. The Web Service paradigm (Kreger, 2001; Gottschalk et al., 2002) is one realization of the Service Oriented Architecture.

With Web services, distributed applications can be encapsulated as self-contained, discoverable and Internet-accessible software components that can be integrated to create other applications. Web services are characterized by a precise separation of service interface description, implementation, and binding, as well as standardized and declarative message-based interactions. The family of specifications that make up the Web service standards includes a specification for composition (Andrews et al., 2003; Kavantzas et al., 2005) of existing services to create new higher-level services. These composition languages define abstractions in the form of workflows (Becker et al., 2002) that represent composite services. The executable abstractions model the interaction between the integrated services. Service composition languages are either general-purpose (Andrews et al., 2003; Kavantzas et al., 2005) or domain-specific, as typically seen in scientific applications (Yu and Buyya, 2005)

The distributed nature of services poses significant problems to building robust service-based applications. Addressing issues concerning dynamic reconfiguration of services collaborating in multiple application scenarios, to make them more dependable, is a challenging task. This is because the integration of multiple services introduces new levels of complexity in management. Due to the autonomy of services involved in a composition, the management of composite services cannot extend into the administrative boundaries of individual services (Vogels, 2003; Dialani et al., 2002). Thus the composed service has no influence over the factors affecting quality of service provision at the level of the atomic service. This contributes to the transiency of services, which may fail due to problems in their environment such as faults and resource starvation. Also, since services interact-

Ezenwoye O., Busi S. and Masoud Sadjadi S.
DYNAMICALLY RECONFIGURABLE DATA-INTENSIVE SERVICE COMPOSITION.
DOI: 10.5220/0002852101250130
In *Proceedings of the 6th International Conference on Web Information Systems and Technology (WEBIST 2010)*, page
ISBN: 978-989-674-025-2

ing in a composition are distributed, there is the added problems concerning the unmanaged nature of the Internet.

The increasing use of software as a service (Zhang et al., 2004; Benslimane et al., 2008; Armbrust et al., 2009; Grossman, 2009) has also lead to increasingly complex and data-intensive service aggregations. The highly available nature of some of these applications demands that they remain operational and rapidly responsive even when failures disrupt some of the nodes in the system (Birman et al., 2004; Deelman and Gil, 2006; Lindholm, 2007). Thus, there is a need to deliver reliable service compositions with attributes that cover functional correctness, performance and dependability (Ouzzani and Bouguettaya, 2004; Gil et al., 2007), especially since current Web services standards provide limited constructs for specifying exceptional behavior and recovery actions. It is difficult for developers of composite services to anticipate and account for all the dynamics of such interactions. This is especially true for data-intensive service compositions where data movement and management is particularly problematic. There is therefore a need for service compositions that are dynamically reconfigurable to make them more robust and dependable. Addressing these challenges calls for the development of novel techniques to modeling and management of composite services.

In this paper, we identify the some key requirements of adaptive data-intensive service composition. We also propose a model of service composition that supports dynamic reconfiguration. The rest of this paper is structured as follows. Section 2 provides an overview of service composition models. In Section 3 we discuss the requirements for dynamically reconfigurable service composition. Finally, some concluding remarks are provided in Section 4.

2 COMPOSITION MODELS

Composite services are an aggregation of two or more services. The composite service is modeled as a graph where the nodes represent tasks and the edges represent some composition constraints in the form data flow or flow control dependencies between tasks. Figure 1 shows the object-relational model of service composition. In this model, the composite service is an aggregation of tasks and the tasks may be implemented by the integrated services. The composite service itself is a service.

The composite services are described and published as service aggregators which are service providers themselves. The service aggregates model

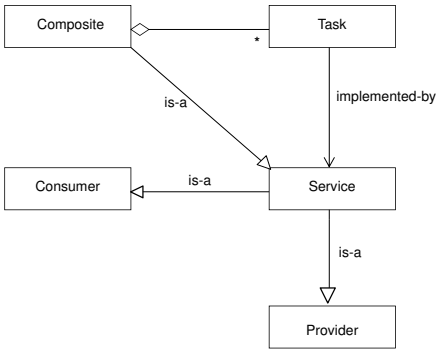

Figure 1: Object relationship for service composition.

and coordinate the interactions between the constituent services. The two main service composition models are orchestration and choreography (Peltz, 2003). In orchestration, the interaction between services is coordinated in a centralized manner by a process with captures the logic of the interaction between the services. The logic of the interaction (control and data dependencies) is encapsulated within the composite service in the form of a workflow. The workflow is executed in an orchestration engine which exposes the workflow (composite) as a service. As illustrated in Figure 2, the order of execution of the integrated services is controlled by the composite via control messages (invocations). Data interchange between the integrated services also passes through the composite. This model presents some advantages in that management of the integration is centralized, however this centralization presents significant challenges to data-intensive compositions and scalability (Liu et al., 2002)

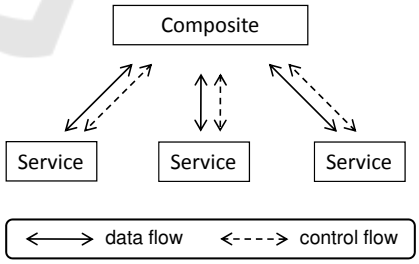

Figure 2: Service Orchestration Model.

In contrast to the orchestration model, service choreography (Figure 3) involves two or more services participating to provide a service, in peer to peer manner. This model is characterized by decentralized control and data flow. Here, each service is aware of its role in the interaction and its immediate partner. This presents significant advantages to scalability and more efficient data flow. However, this model is difficult to manage especially in exceptional conditions.

Dynamic reconfiguration is difficult since no service has a global view of the overall integration.

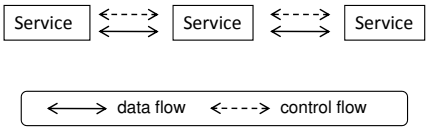

Figure 3: Service Choreography Model.

To overcome some of the limitations of orchestration, some service orchestration platforms do employ decentralized data flow techniques, as illustrated in Figure 4. This approach is typically utilized for data-intensive composition which is often seen in scientific workflow application. In this model, integrated services also encapsulate data movement and management logic to facilitate the movement of data in a peer-to-peer manner. Control flow logic is retained by the composite thus issues with scalability persist. Also the composite service constitutes a central point of fail. Besides, dynamic reconfiguration is still a problem since the composite cannot easily be modified to react to changes in the environment of the interaction.

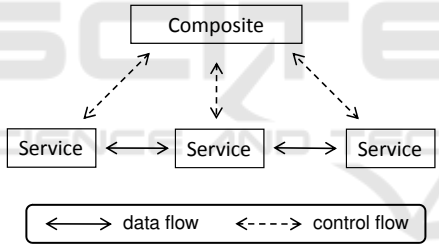

Figure 4: Service orchestration with decentralized data flow.

3 REQUIREMENTS FOR ADAPTABILITY

Good examples of data-intensive applications are those found in the domain of scientific workflows (Jaeger et al., 2005; Ezenwoye et al., 2007). Here service integration is achieved through composite services which are used to create applications for coordinated problem solving. Scientific applications are both data intensive and computational intensive. These applications require a number of systems working together, accessing large amounts of data and analyzing the data in parallel. Scientific applications analysis takes long time to complete. Thus it is necessary that the application runs successfully in spite of changes in execution and the execution environment. The adaptability of the architecture of such applications is required to enable reconfiguration of the system. Below we identify the major requirements for adaptive service composition.

3.1 High-level Composition

Key to the realization of the benefits of service-oriented computing is the ability to integrate basic services to create higher-level applications. These higher-level applications will provide the right level of abstraction for the non-computer scientists. Thus, allowing them to concentrate on their domain specific work instead of the technical issues of underlying layers. Workflow languages permit such aggregation of services to create coarse-grained processes that constitute a number of related functions. With such languages, higher-level application can be modeled as graphs where the nodes represent tasks while the edges represent inter-task dependencies, data flow or control flow.

Although complex processes can be developed with general-purpose languages such as Java and C++, such languages do not provide high-level constructs to easily define processes that represent composite services. High-level composition tools make it possible for domain specialist to deploy flexible service-oriented applications. However, current composition tools are too cumbersome and complex to allow for easy service composition. High-level tools need to have support for separation of concerns that would permit aspect-oriented (Kiczales et al., 1997) techniques, so that aspect weaving capabilities that address quality of service concerns can be introduced. This is important so that users can for instance, specify failure handling policies at a higher level of abstraction without having to entangle them within the application logic.

For higher level service compositions there are many aspects that need to be incorporated in order to provide more flexibility, adaptability and reliability in the system. Some other requirements for higher-level composition are to provide coordination methods, dynamic models and automated service integration and composition (Papazoglou et al., 2007; Ezenwoye and Sadjadi, 2006).

3.2 Extensible Recover Mechanisms

Due to the heterogeneous nature of the application context, failure recovery mechanisms should support a wide range of failure handling strategies including user-defined exception handling. Flexible failure handling mechanism should allow multiple fault

tolerance techniques and strategies (e.g., retry, migration, restart and checkpointing) This would permit each task to select an appropriate fault tolerance technique among alternatives depending on the characteristics of the task or the underlying execution environment (Hwang and Kesselman, 2003) Support for extensible failure handling mechanisms should be provided so that it is possible to specify at a high-level, such diverse failure handling strategies within a recovery policy that can be modified at runtime. Such an approach would permit a user to define specific exceptions and exception handling since the recovery mechanism is extensible. However, such mechanisms should not prevent the developer of the application from specifying specific exception handling. Tasks for which exception handling has been defined within the application, may be left untouched by the adaptation mechanism, if so desired.

3.3 Robust Data Management

The inherent nature of the infrastructure and environment for distributed composite applications means that the management of data comes with challenges. The successful execution of applications is dependent on the availability of necessary data. However, moving an application close to the data may not always be practical due to insufficient computational resources at the storage site (Kosar and Livny, 2005), so data needs to be moved to the applications that need them and in some cases cleanup operations are required after application execution. Also, deployed composite services may be unable to execute due to the lack of the disk space, requiring that data movement be scheduled and monitored. Thus, the management of data is essential through the entire lifecycle of such distributed applications from creation to execution, and result management (Deelman and Chervenak, 2008)

Although data is a key component, a lot of emphasis is not placed on providing fault tolerance for tasks related to data requirements. For instance, data staging tasks are often embedded in computation-related tasks and reliability efforts are then focused on the computation tasks even though data access presents the main bottleneck for data-intensive applications (Kosar and Livny, 2005) Thus improving the reliability calls for the decoupling of data staging and computation activities, and each aspect needs to be addressed separately (Ranganathan and Foster, 2002; Ezenwoye et al., 2009) Also, infrastructure for distributed data-intensive composite services needs to consider data movement as part of the end-to-end performance of the system. Techniques for dynamic scheduling, caching and data replication should be used to ensure availability of date. Care must be taken to make sure they complete successfully and without any need for human intervention (Kosar and Livny, 2005) Since services are often composed by domain specialists that are not particularly familiar with performance and fault-tolerance issues of the underlying layer, constructs for data management need to be separated from business logic of the composite application during development time.

3.4 Modularization

Data-intensive service compositions have a tendency to be large and complex with multiple cross-cutting concerns. Facilitating reconfiguration calls for modularization of composite services into pluggable pieces that encapsulate various concerns. For instance, recovery mechanisms should enable the separation of failure handling policies from application codes. This requirement is driven by the dynamic nature of the environment, and would permit the adaptation of failure handling policies to changing environments by modifying the high-level policy description. Recovery policy specification, monitoring, exception handling and data management logic should be modular and externalized to keep the service composition uncluttered. This separation also ensures that components are able to evolve separately.

Also, coding manually various cross-cutting concerns within the application is not a viable solution because it makes the development much more complicated. Therefore, generative programming techniques should be used to apply software patterns (Buschmann et al., 2007) which constitute abstract reusable concepts that can be configured for a range of situations. The reuse of these abstract concepts is facilitated by the fact that applications in the same domains are similar and carry out comparable functions (Sommerville, 2006).

3.5 A Hybrid Composition Model

In this Section we propose a model of service composition to support dynamic reconfiguration. This model is a combination of orchestration and choreography models, and overcomes some of the limitations orchestration and choreography. The model has a composite service, which we will referred to as the monitor (Figure 5). The monitor is a composite which models the logic of the interaction between the services participating in the choreography. Thus the monitor has the global view of the interaction between the interacting services. This component mon-

itors the interactions and intervenes only if necessary to provide required adaptability to the system. This monitor node observes the data and control-flow of the choreography. In case of failure, the monitor intervenes and for instance, assigns the load to another service and provides it with the information to resume the task to be performed. This monitor would encapsulate the behavioral and recovery policies that allow for the choreography to be reconfigured.

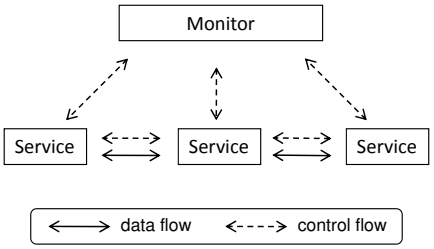

Figure 5: A hybrid service composition model.

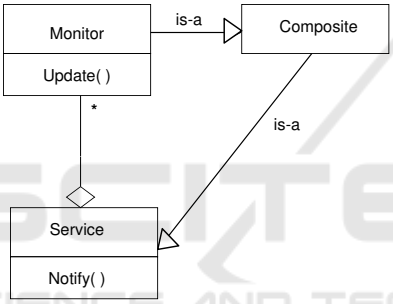

Figure 6: Object relationship for hybrid composition.

Figure 6 shows the object-relational model for hybrid composition. The data stored by the monitor reflects the state of the choreography, therefore the relationship is the Observer pattern. The Observer design pattern allows for dependency between object to be specified so that when one component changes state all dependent components are notified and updated (Gamma et al., 1995) This pattern of behavior is especially important in distributed environments where services are statefull and long-running. Services provide notification interfaces that allow for observers to be updated about state change. In Figure 6, Monitor is the observer and registers for state change notification with the monitored services in the choreography. Note that the observer in this case does not constitute a single point of failure since multiple observers can be used.

4 CONCLUSIONS

In this paper, we presented the orchestration and choreography service composition models, and pointed out their limitations. In orchestration, a central point of control for both control messages and data exchange becomes a bottleneck for data intensive applications. While in choreography, the lack of global view of the system fails to incorporate dynamic reconfiguration in the system.

We identified the requirements for dynamically reconfigurable service compositions for data intensive applications. A hybrid composition model that combines the positive attributes of orchestration and choreography is also present. The hybrid model of service composition supports the dynamic reconfiguration for data-intensive applications. This model includes a monitoring node that has a global view of the choreography and observe the behavior of services involved in choreography. Reconfiguration is achieved through the monitoring component.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under Grant No. OISE-0730065. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Andrews, T., Curbers, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thattee, S., Trickovic, I., and Weerawarana, S. (2003). Business process execution language for web services.

Armbrust, M., Fox, A., Joseph, R. G. A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Becker, J., Gille, M., Muehlen, M. Z., and Gille, D. M. (2002). Workflow application architectures: Classification and characteristics of workflow-based information systems. In *in: Fischer, L. (Ed.): Workflow Handbook 2002, Future Strategies, Lighthouse Point, FL*, pages 39–50.

Benslimane, D., Dustdar, S., and Sheth, A. (2008). Services mashups: The new generation of web applications. *IEEE Internet Computing*, 12(5):13–15.

Bichler, M. and Lin, K.-J. (2006). Service-oriented computing. *Computer*, 39(3):99–101.

Birman, K. P., van Renesse, R., and Vogels, W. (2004). Adding high availability and autonomic behavior to web services. In *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, pages 17–26, Edinburgh, United Kingdom. IEEE Computer Society.

Buschmann, F., Henney, K., and Schmidt, D. C. (2007). *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Volume 4*. Wiley.

Deelman, E. and Chervenak, A. (2008). Data management challenges of data-intensive scientific workflows. In *Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid*, Washington, DC, USA. IEEE Computer Society.

Deelman, E. and Gil, Y. (2006). Managing large-scale scientific workflows in distributed environments: Experiences and challenges. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 144, Washington, DC, USA. IEEE Computer Society.

Dialani, V., Miles, S., Moreau, L., Roure, D. D., and Luck, M. (2002). Transparent fault tolerance for web services based architectures. In *Eighth International Europar Conference (EURO-PAR'02)*, Lecture Notes in Computer Science, Padeborn, Germany. Springer-Verlag.

Ezenwoye, O. and Sadjadi, S. M. (2006). Composing aggregate web services in BPEL. In *Proceedings of The 44th ACM Southeast Conference*, Melbourne, Florida.

Ezenwoye, O., Sadjadi, S. M., Carey, A., and Robinson, M. (2007). Grid service composition in bpel for scientific applications. In *Proceedings of the International Conference on Grid computing, high-performAnce and Distributed Applications*, Vilamoura, Algarve, Portugal.

Ezenwoye, O., Viswanathan, B., Sadjadi, S. M., Fong, L., Dasgupta, G., and Kalayci, S. (2009). Task decomposition for adaptive data staging in workflows for distributed environments. In *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering*, pages 16–19, Boston, MA.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company.

Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., and Myers, J. (2007). Examining the challenges of scientific workflows. *Computer*, 40(12):24–32.

Gottschalk, K. D., Graham, S., Kreger, H., and Snell, J. (2002). Introduction to web services architecture. *IBM Systems Journal*, 41(2):170–177.

Grossman, R. L. (2009). The case for cloud computing. *IT Professional*, 11(2):23–27.

Hwang, S. and Kesselman, C. (2003). Gridworkflow: A flexible failure handling framework for the grid. In *Proceedings of The 12th IEEE International Symposium on High Performance Distributed Computing*, Seattle, Washington.

Jaeger, E., Altintas, I., Zhang, J., Ludscher, B., Pennington, D., and Michener, W. (2005). A scientific workflow approach to distributed geospatial data processing using web services. In *Proceedings of the 17th international conference on Scientific and statistical database management*, pages 87–90. Lawrence Berkeley Laboratory.

Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y., and Barreto, C. (2005). *Web Services Choreography Description Language*. W3C, 1.0 edition.

Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag.

Kosar, T. and Livny, M. (2005). A framework for reliable and efficient data placement in distributed computing systems. *Journal of Parallel and Distributed Computing*.

Kreger, H. (2001). *Web Services Conceptual Architecture*. IBM Software Group.

Lindholm, K. R. (2007). The user experience of software-as-a-service applications. Technical Report UCB iSchool Report 2007-005, The School of Information, University of California, Berkeley.

Liu, D., Law, K. H., and Wiederhold, G. (2002). Analysis of integration models for service composition. In *Proceedings of the 3rd international workshop on Software and performance*, pages 158–165. ACM.

Ouzzani, M. and Bouguettaya, A. (2004). Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44.

Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45.

Peltz, C. (2003). Web services orchestration and choreography. *IEEE Computer*, 36(10):44–52.

Ranganathan, K. and Foster, I. (2002). Decoupling computation and data scheduling in distributed data-intensive applications. In *International Symposium on High-Performance Distributed Computing*, Los Alamitos, CA, USA. IEEE Computer Society.

Sommerville, I. (2006). *Software Engineering 8th edition*. Addison-Wesley.

Vogels, W. (2003). Web services are not distributed objects. *IEEE Internet Computing*.

Yu, J. and Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49.

Zhang, L.-J., Li, H., and Lam, H. (2004). Services computing: Grid applications for today. *IT Professional*, 6(4):5–7.