Semi-automatic Dependency Model Creation based on Process Descriptions and SLAs

Matthias Winkler[1], Thomas Springer[2], Edmundo David Trigos[1] and Alexander Schill[2]

[1] SAP Research CEC Dresden, SAP AG, Chemnitzer Str. 48, 01187 Dresden, Germany

[2] TU Dresden, Faculty of Computer Science, Institute for Systems Architecture Computer Networks Group, Nthnitzer Str. 46, 01187 Dresden, Germany

Abstract. In complex service-oriented business processes the composed services depend on other services to contribute to the common goal. These dependencies have to be considered when service compositions should be changed. Information about dependencies is only implicitly available from service level agreements and process descriptions. In this paper we present a semi-automatic approach to analyze service dependencies and capture information about them explicitly in a dependency model. Furthermore, we describe a system architecture which covers the whole process of dependency analysis, dependency model creation and provisioning. It has been implemented based on a healthcare scenario.

1 Introduction

According to the Internet of Services vision services traded via open marketplaces are composed to business processes. Services are provided fully automatically (credit card check) or involve manual steps (healthcare services). The composed services have to collaborate to achieve a common goal. Thus, the composition creates different types of dependencies between involved services, e.g. with respect to produced and consumed resources, timing, quality of service (QoS), and pricing. Dependencies occur between atomic services (horizontal dependency) or between atomic services and the composition (vertical dependency).

Explicit knowledge about dependencies is needed for the management of service level agreements (SLA) in service compositions. SLAs are negotiated between the service provider and consumer to regulate service provisioning. During the negotiation process it is necessary to ensure that all SLAs of the composition enable the proper collaboration between the different services and the fulfillment of all SLAs. Furthermore, dependency information is also needed for the handling of SLA violations or explicit SLA renegotiation requests by the different stakeholders. SLA violations as well as the renegotiation of SLAs may affect other services and lead to the violation of other SLAs. Thus, information about service dependencies is needed for SLA management by composite service providers. Required information about service dependencies is usually not explicitly available but is implicitly contained in SLAs and process descriptions. From these sources it has to be extracted to be available at runtime.

DOI: 10.5220/0003042900160028

In previous work we presented an approach to managing dependencies in service compositions [1], where dependencies are analyzed at design time, dependency information is captured in a dependency model [2], and dependency information is used at runtime to evaluate effects of SLO violations and SLA renegotiation requests on other services. This paper extends our work by two major contributions. Firstly, we detail our work on dependency model creation by a process description. Secondly, we present an architecture for managing dependencies. The remainder of this paper is structured as follows: In Chapter 2 we describe the dependency model creation process followed by the presentation of the architecture of the approach in Chapter 3. We evaluate our work in Chapter 4 and discuss it with respect to related work in Chapter 5. Finally, we conclude this paper with a discussion and outlook on future work in Chapter 6.

2 Dependency Model Creation

In order to manage the dependencies between services throughout the lifecycle of a composite service, we developed an approach which captures dependencies in a dependency model at design time and which uses this information to evaluate effects of SLO violations as well as SLA renegotiation requests at runtime. We developed a lifecycle for managing and using dependency information. This lifecycle consists of four phases: creation and recalculation, validation, usage, and retirement. In this chapter we will detail the first lifecycle phase focusing on the model creation and its integration into the development process of the composite service. In Fig. 1 the dependency model creation process is depicted. This semi-automatic process is initiated by the composite service modeler. As pre-requisite for executing the process two aspects have to be fulfilled:

1. The composite service workflow has been modeled (e.g. BPMN process). It provides information on temporal relationships between services.
2. SLA offers for all services are available specifying information regarding execution time and location, handled resources, supported QoS, and service price. This information is needed for dependency model creation.

As a result a dependency model is created, which still needs to be validated with respect to the negotiated SLAs. This is necessary in order to avoid conflicts between the proposed SLAs (e.g. with respect to time and QoS attributes).

The creation process of a dependency model was realized as a semi-automatic approach consisting of automatic dependency discovery and the explicit modeling of dependencies. The discovery of dependencies automates part of the dependency model creation process. It also helps to reduce the chance of errors such as false or missing dependencies introduced by manual modeling. The manual extension and modification of the generated dependency model enables the expression of dependencies which cannot be discovered automatically. However, it also introduces the chance of errors being added to the dependency model. In the following sections the dependency discovery and dependency modeling are explained in more detail.

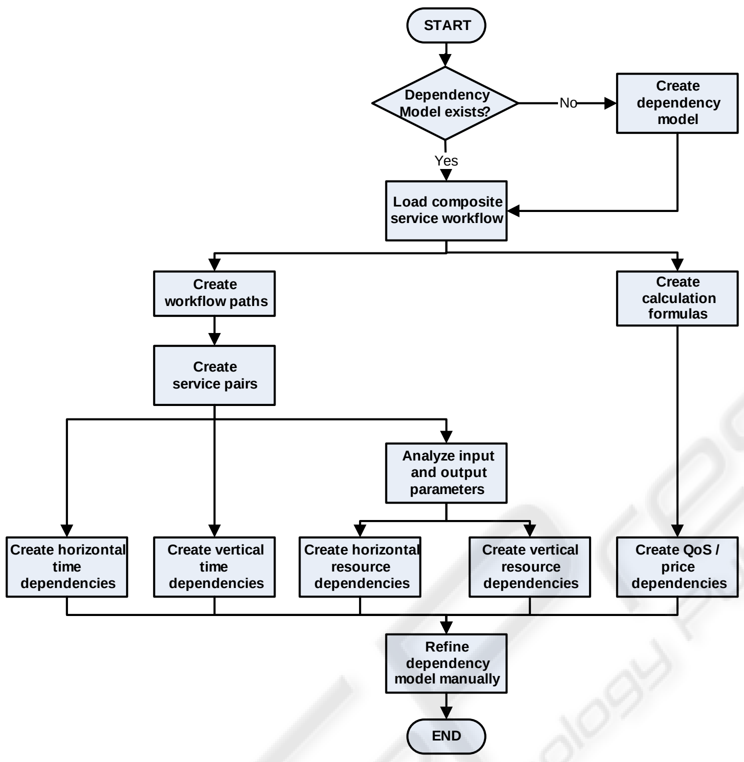

START
Dependency Model exists?
No
Yes
Create dependency model
Load composite service workflow
Create workflow paths
Create calculation formulas
Create service pairs
Analyze input and output parameters
Create horizontal time dependencies
Create vertical time dependencies
Create horizontal resource dependencies
Create vertical resource dependencies
Create QoS / price dependencies
Refine dependency model manually
END

Fig. 1. Dependency model creation process.

2.1 Dependency Model Creation Process

As a first step in the process of creating a dependency model a new model instance is created if it does not already exist. After that two parallel tasks are started for the discovery of dependencies. This includes the creation of time and resource dependencies on the one hand and the creation of aggregation formulas for QoS attributes and price information on the other hand. For the creation of time and resource dependencies the first step is the creation of linear paths reaching from the start node to the end node of the composite service workflow. For each path pairs of services are created. The selection of the relevant services for pair creation is dependent on the type of dependency which is analyzed (see section 2.2). Based on the created pairs the analysis of dependencies is done. The creation of time dependencies is directly based on the different pairs. No further analysis is necessary, since time dependency creation is based on the process structure only, i.e. if a service $S2$ follows service $S1$ in the process, this implies that $S1$ is executed before $S2$. For the creation of resource dependencies the input and output parameters of two services are compared. If a match is found, a resource dependency is created. The different dependencies are then added to the dependency model. The analysis for QoS and price dependencies is based on [3]. It starts with a reduction of the service workflow based on workflow patterns. Formulas for calculating composite QoS and price values for the respective workflow patterns are selected and an aggregation formula is created. Finally, a dependency is created for each composite QoS and price value. Following the discovery of dependencies the created dependency model can be refined in a manual modeling step.

2.2 Dependency Discovery

The goal of our work is to provide composite service providers with information about service dependencies in a composition. This information should facilitate the management of SLAs. A SLA contains a list of parameters such as time, price, location, and quality of service provisioning. Dependencies occur with regard to these parameters: e.g. the composite service price depends on the atomic service prices; provisioning of the first atomic services in the composition can be started as soon as the provisioning of the composite service is initiated; provisioning of two atomic services needs to be started or finished at the same time. These brief examples show that different types of dependencies exists and that fine grained dependency information is required.

The discovery of dependencies is specific for each dependency type. We will now describe the discovery of time and resource dependencies in more detail including the selection of services for service pair creation as well as the type of dependency being created. Time dependencies are expressed based on time relations as used in project management [4] or as defined by Allen [5]. For resource dependencies the different resources are listed. An overview of the mappings for creating dependencies is presented in Table 1. The creation of aggregation formulas for composite QoS and price values is achieved as described by [3]. Due to space limitations we would like to point the reader to the respective work for further information.

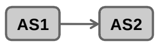
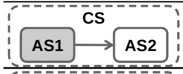
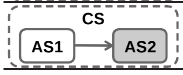
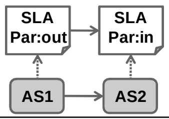
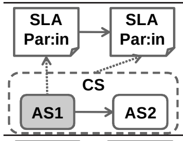
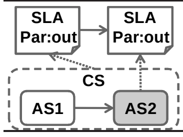
Horizontal Time Dependencies are created between each pair of services, where one service is directly connected to another service in a path. For each pair a *finish-to-start* time dependency is created between the earlier and the later service.

Vertical Time Dependencies are created between the composite service and the first and last atomic service within a path. Between the composite service and the first atomic service a *start-to-start* time dependency is created. Between the last atomic service and the composite service a *finish-to-finish* time dependency is created.

Horizontal Resource Dependencies are created between atomic services, which are directly or indirectly connected within a path. To check whether two services have a dependency the output of the preceding service is compared to the input of the succeeding service. If a match is found a resource dependency is created. Information on input and output of services is available from their SLAs.

Vertical Resource Dependencies are created between the composite service and an atomic service. For each path the composite service input and output is compared to the atomic services input and output. A resource dependency is created with all atomic services along a path, which have a matching input with the composite service input and which do not have a horizontal dependency regarding the matching resources. A further resource dependency is created with the last atomic service, which has a matching output with the composite service.

Table 1. Comparison of dependency model approaches.

Composite service construct	**Description**	**Dependency model construct**
	Two atomic services directly connected via control flow	Time dependency: finish-to-start
	Composite service and first atomic service in a path	Time dependency: start-to-start
	Last atomic service and composite service in a path	Time dependency: finish-to-finish
	Output of preceding atomic service matches input of succeeding service	Resource dependency: AS2.paramIn resourceDependent AS1.paramOut
	Input of composite service matches input of atomic service	Resource dependency: AS1.paramIn resourceDependent CS.paramIn
	Output of composite service matches output of atomic service	Resource dependency: CS.paramOut resourceDependent AS2.paramOut

2.3 Dependency Modeling

The dependency discovery algorithm produces a valid dependency model. However, there are several types of dependencies which cannot be discovered. This includes dependencies regarding the location for executing a service or QoS dependencies where no aggregation formula can be created automatically. Furthermore, time dependencies may exist between services which are not connected by the process flow. A concrete use case may have time constraints, which have to be modeled explicitly, i.e. the created dependency model is extended manually.

3 Architecture and Integration

In this chapter the architecture of the dependency management components as well as their integration into a service engineering toolchain are described. The components provide functionality for the creation, validation, and storage of dependency models (*Dependency Model Management*), the analysis and modeling of dependencies (*Dependency Analysis*), and the evaluation of the dependency model with respect to different events at runtime (*Runtime Dependency Evaluation*). An overview of the components is presented in Fig. 2. Details about their functionality are described below.

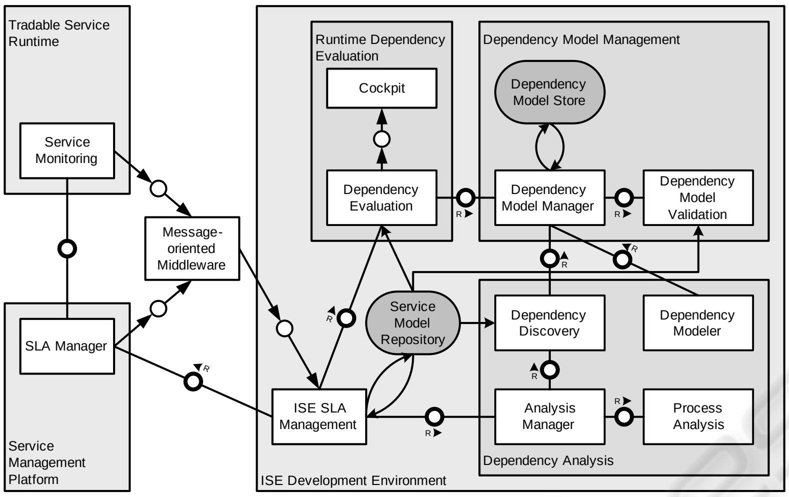


Fig. 2. Architecture Dependency Handling.

3.1 Dependency Model Management

The approach for managing service dependencies has at its core the dependency model, which is used to capture information about services and the dependencies that occur between them. The components, which are part of the dependency model management, are responsible for the creation, validation, and storage of dependency models and for making these models available to other components.

The *Dependency Model Manager* is the central component. It creates new dependency model instances for each new SLA negotiated for a composite service. It is also responsible for adding information to dependency models and making model information available to other components such as *Dependency Modeler* and *Dependency Evaluation*. The *Dependency Model Validation* component is responsible for validating the dependency model with respect to the defined constraints and the respective SLAs. An example is the validation of negotiated times which are discovered based on the workflow structure or modeled manually. Furthermore, validation regarding more general aspects is realized (e.g. each consumed resource needs to be provided by an entity). The final dependency model instances are stored in the *Dependency Model Store*.

3.2 Service Dependency Analysis

The analysis of dependencies is executed after creating the service composition and during the process of negotiating SLAs for the different services. It requires a process description and SLAs in the offer state (i.e. containing offered SLO values) as input. Our implementation is based on BPMN (Business Process Modeling Notation) process descriptions. BPMN represents a suitable means for modeling business processes from a business perspective at an abstract level. An alternative approach would be the usage of a BPEL (Business Process Execution Language) process notation. BPEL is, however, targeted at processes that are executed automatically and which are realized by web services. Processes involving mainly human or machine tasks are typically not modeled using BPEL. The dependency analysis functionality is distributed between components supporting the automatic dependency discovery as well as dependency modeling. The analysis process and the involved components are presented in Fig. 3.

The *Analysis Manager* handles the process of dependency discovery. It is initiated by the composite service creator during the negotiation of SLAs with the consumer of the composite service as well as the atomic service providers. It retrieves the workflow description and SLA documents for the analysis and initiates the different steps of the discovery. The *Process Analysis* component is responsible for decomposing the process into linear paths leading from the start node all the way to the end node. These paths are used for the discovery of dependencies. The *Dependency Discovery* component realizes the different dependency discovery mechanisms. They include all horizontal and vertical dependency evaluation tasks as described in section 2.2. The implementation is based on the generated paths and SLA information. When it discovers a dependency it requests the *Dependency Model Manager* to add the respective information to the dependency model. Once the information has been added to the model, it is stored in the *Dependency Model Store*. From there it can be accessed for further handling.

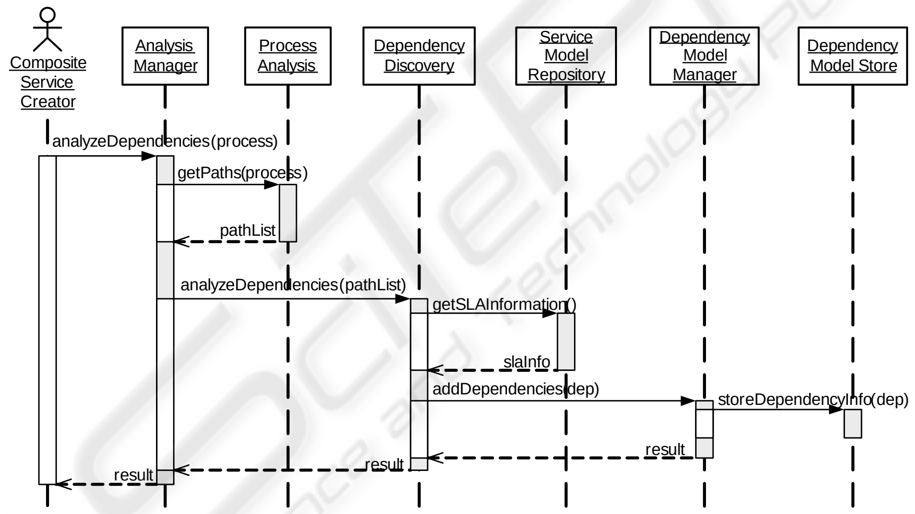


Fig. 3. Process and components for analysis of dependencies.

The *Dependency Modeler* provides dependency modeling functionality for the composite service creator. It enables the creation of new as well as the adaptation of existing dependency models. It was realized as a graphical model editor. The modeling process is initiated by the composite service creator. As a first step the *Dependency Modeler* requests a dependency model from the *Dependency Model Manager*. Once the model is available, the composite service creator uses the editing functionality to add, remove, or modify dependency and service information in the dependency model.

3.3 Runtime Dependency Evaluation

The *Runtime Dependency Evaluation* component is responsible for the evaluation of dependency information at runtime. The occurrence of SLO violation information requires the determination of effects of this violation on other services (atomic or composite service). Requests for renegotiating an SLA need to be evaluated with regard to effects on other services before accepting them.

The runtime evaluation of dependencies is initiated by the *ISE SLA Management* component calling the *Dependency Evaluation* component which executes the evaluation process. It requests the relevant dependency model from the *Dependency Model Manager* and evaluates it. Since the runtime dependency evaluation is not in the focus of this paper we do not present more details about this.

3.4 Integration with Service Engineering Toolchain

The different dependency management components are integrated into the ISE development environment, a tool created for the modeling of services. This enables proper handling of dependencies for composite service providers while modeling their services. Within the ISE development environment the dependency analysis components also have access to the necessary information for executing the analysis (i.e. the BPMN process description and SLA information). The *Dependency Modeler* tool is also integrated into the ISE development environment. The *Runtime Dependency Evaluation* component is integrated with the *ISE SLA Management* components, which handle the integration with the *Service Monitoring* on the *Tradable Service Runtime* (TSR) and the *SLA Manager* on the *Service Management Platform* (SMP) respectively. The TSR provides the service runtime infrastructure while the SMP offers service marketplace functionality.

4 Evaluation

In the first part of the evaluation we discuss the performance of the algorithms used for the automatic discovery of dependencies. In the second part we present a set of test cases to better illustrate the different steps of the dependency analysis process. The results of both parts are discussed in a third section.

The performance measurements and test case handling were executed using the workflow of a composite healthcare service (see Fig. 4). The scenario is based on a healthcare workflow presented in [6]. In this scenario a patient undergoes several examinations at a healthcare center. The different examinations are executed by different medical service providers. Further services include the analysis of blood samples, creation of documentation, and transport of the patient.

4.1 Performance Considerations

As a first step we measured the times taken for the different tasks of the dependency discovery approach for the healthcare service: path creation (5 ms), horizontal (7 ms)

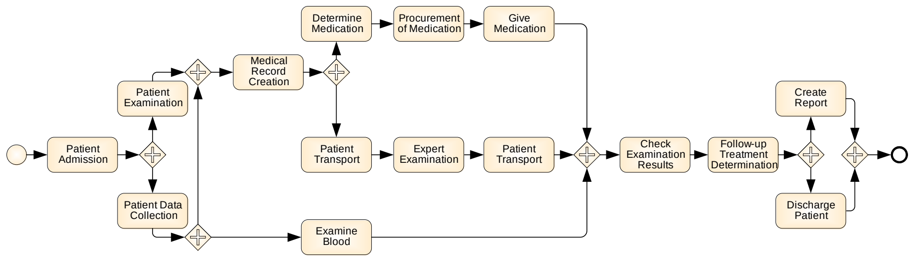


Fig. 4. Workflow of composite service - Stationary Patient Check-up.

and vertical (2 ms) time dependencies, horizontal (3 ms) and vertical (25 ms) resource dependencies. The results show that all tasks are executed within a few milliseconds.

Furthermore, a number of measurements were made to test the scalability of the approach. We modeled 5 business process workflows (P1..P5) of different complexity (number of nodes, number of splits and joins). We counted the number of artifacts created during dependency analysis and measured the times for creating relevant service pairs. The results presented in Table 2 show that with increasing workflow complexity the number of paths as well as duplicate time and resource pairs increases strongly. Thus it is necessary to ensure that the further analysis of pairs is only executed for pairs which have not been handled before. The results also show that the discovery of dependencies in relatively complex services is executed in less than a second.

Table 2. Measurement results.

	P1	**P2**	**P3**	**P4**	**P5**
Nodes	12	31	36	55	87
Split/join	1/5	13/12	28/9	17/32	94/44
Created paths	21	60	952	1933	908
Duplicate time pairs	76	102	7387	14554	5535
Non-duplicate time pairs	14	32	45	74	91
Duplicate resource pairs	222	139	33686	65569	20951
Non-duplicate resource pairs	57	103	422	679	916
Time to get time pairs (ms)	1.7	2.0	78.7	289.7	293.8
Time to get resource pairs (ms)	0.9	2.1	188.7	554.0	153.5

4.2 Test Case based Evaluation

A number of test cases serve as the base for validating the different steps of the dependency analysis process. Each test case is illustrated with a brief example. For the analysis of dependencies the service workflow (see Fig. 4) and SLA descriptions are needed. Due to space limitations only excerpts of SLAs of services relevant for the presented examples are listed in Table 3.

TC1 - Path Creation: The composite service workflow is decomposed into linear paths. *Results:* List of 10 paths reaching from the start to the end of the process. One example path is the following: *Patient Admission - Patient Data Collection - Examine Blood - Check Examination Results - Follow-up Treatment Determination - Create Report*

Table 3. Sample SLA information.

Service	Input resources	Output resources
Patient Admission	-	patient ID
Examine Blood	patient ID, blood sample	laboratory test result
Create Report	medical record	examination report
Stationary Patient Check-up	-	examination report

TC2 - Horizontal Time Dependencies: Pairs of directly connected services are created along the paths. Duplicate pairs (e.g. pair *Follow-up Treatment Determination - Create Report* occurs in 5 paths) are removed. Time dependencies of type *finish-to-start* are created for each pair. *Results:* List of horizontal time dependencies. One horizontal time dependency between *Patient Data Collection* and *Examine Blood* is shown in Table 4.

TC3 - Horizontal Resource Dependencies: All pairs of different services along the paths are created. Duplicate pairs are removed. All pairs are analyzed with regard to matching input and output resources. Information about input and output resources is taken from the negotiated SLAs (see Table 3). Resource dependencies are created when matching resources are found. *Results:* List of horizontal resource dependencies. One resource dependency between the service *Patient Admission* and *Examine Blood* is shown in Table 4.

TC4 - Vertical Time Dependencies: Creation of vertical time dependencies between the composite service and the first (*start-to-start*) and last (*finish-to-finish*) atomic services in the paths. *Results:* List of vertical time dependencies. One vertical time dependency between *Stationary Patient Check-up* and *Patient Admission* is shown in Table 4.

TC5 - Vertical Resource Dependencies: All atomic services along the paths are analyzed with regard to matching input and output resources with the composite service. Dependencies are created for matching resources if no horizontal dependency exists regarding the matching resources. *Results:* List of vertical resource dependencies. One vertical resource dependency between *Create Report* and *Stationary Patient Check-up* is shown in Table 4.

TC6 - Dependency Model Extension: Manual creation of a location dependency between the *Patient Transport* service and the *Expert Examination* service. *Results:* One location dependency between *Patient Transport* and *Expert Examination* (see Table 4).

4.3 Discussion

In this chapter we demonstrated the general feasibility of the approach to create dependency models. We first presented an overview about performance measurements. One result of the measurements was that with increasing complexity of the workflows not

Table 4. Dependencies of healthcare process.

Antecedent - Dependant	Dependency	Description
Patient Data Collection - Examine Blood	time	endTime *finish-to-start* startTime
Patient Admission - Examine Blood	resource	patient ID
Stationary Patient Check-up - Patient Admission	time	startTime *start-to-start* startTime
Create Report - Stationary Patient Check-up	resource	examination report
Patient Transport - Expert Examination	location	endLocation *equals* startLocation

only the number of paths and relevant service pairs increased, but that a proportionally large amount of time would be necessary for handling duplicate services. Thus, they need to be removed. However, we also showed that the time needed to analyze relatively complex processes is still less than a second, which allows to use the approach at design time.

As a second part of the evaluation we applied different test cases which demonstrated the functioning of our approach. We showed a sample path created during the execution of the dependency discovery as well as a number of different dependencies. In total this scenario produces 40 time and resource dependencies, which are discovered automatically. Two more location dependencies can be modeled. A manual handling of all these dependencies would be very time consuming and error prone. In more complex processes the number of dependencies will be much higher, which renders a manual handling of dependencies even more difficult. Our semi-automatic process facilitates this.

5 Related Work

The handling of dependencies between services has been addressed for a variety of purposes including the automatic composition of services [7], the optimization of sequencing constraints of composite services ([8, 9]), root cause and impact analysis ([10, 11]), and SLA management ([3, 12]).

Wu et al. [8] present an approach for modeling and optimizing the synchronization dependencies of activities in business processes. A synchronization model, which contains dependency information, is used to support activity scheduling in business processes. In contrast to our approach automatic discovery of dependencies is not supported. In [9] the authors discuss control and data dependencies in business processes and argue that they form the base for sequencing constraints in business processes. They present an approach for deriving control dependencies from semantically annotated business activities by evaluating their pre-conditions and effects. Input and output parameters of business activities form the base for data dependencies. This approach differs from our approach in several ways: While our resource dependencies are similar to the data dependencies of their work, we also support dependencies regarding time, location, QoS, and pricing information. Furthermore, their approach is limited to dependencies between atomic services while our work also supports dependencies between atomic and composite services.

Ensel and Keller [10] introduce an approach to handle dependencies between managed resources (e.g. web application server, database, operating system) in a distributed system. The goal is to support root cause as well as impact analysis for service failure situations. Dependencies are represented in a distributed dependency model which captures the dependencies and attributes of these managed resources. However, no work is presented with regard to the discovery of service dependencies. The MoDe4SLA [11] approach supports the handling of response time and price dependencies of composite services on its atomic services. The goal of the system is to support root-cause analysis for problems caused by atomic services. The dependency information is captured by a modeling approach. The discovery of dependencies is not supported.

The COSMA approach [3] supports the providers of composite services to manage their SLAs. Dependencies between composite QoS values and atomic ones are expressed using aggregation formulas. The aggregation formulas for the different QoS values are automatically derived from the process description. Further constraints need to be added manually or from configuration files. In contrast to our work, COSMA focuses only on the relationship between composite services and atomic services, but dependencies between atomic services are not handled . Dependency types such as resource, location, and time are not covered. However, our approach to QoS and price dependency discovery is based on COSMA. Karnke et al. describe an agent-based approach to managing SLAs in value chains [12]. The focus is on SLA based resource management in hierarchies of service level agreements. As part of the agent-driven negotiation process, dependencies between services are considered. However, no work has been presented regarding the discovery of dependencies.

6 Conclusions

The approach presented in this paper enables management of service dependencies. We have shown that different types of dependencies can occur in parallel within complex service compositions representing business processes. It has been demonstrated that a significant subset of these dependency types can be automatically extracted from information provided by the process description and the SLAs negotiated between the involved service providers and consumers. Based on the different characteristics of service dependencies specific algorithms have been identified for automatic dependency discovery. These algorithms are embedded into the process of dependency management implemented by the presented architecture for dependency handling, particularly in the dependency analysis component. In the process the evaluation of service dependencies at runtime is foreseen. The automatically discovered dependencies are stored in a dependency model and are made available for runtime dependency evaluation. The presented performance measurements prove the applicability of our algorithms for dependency discovery at runtime, since the processing time is in the range of few seconds even for complex processes of up to 100 services. The test case based evaluation demonstrated the feasibility of our approach, illustrating the complexity of the dependency discovery and showing created artifacts.

In the future we will consider additional types of dependencies with respect to characteristics, detection algorithms and modeling. Further use case studies will be carried out to prove the applicability and practical relevance of our approach. Finally, the runtime dependency evaluation will be implemented.

Acknowledgements

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference "01MQ07012". The authors take the responsibility for the contents.

References

1. Winkler, M., Schill, A.: Towards dependency management in service compositions. In Filipe, J., Marca, D.A., Shishkov, B., van Sinderen, M., eds.: ICE-B 2009 - Proceedings of the International Conference on e-Business, Milan, Italy. (2009)
2. Sell, C., Winkler, M., Springer, T., Schill, A.: Two dependency modeling approaches for business process adaptation. In Karagiannis, D., Jin, Z., eds.: Knowledge Science, Engineering and Management, Springer (11 2009)
3. Ludwig, A., Franczyk, B.: Cosma–an approach for managing slas in composite services. In Bouguettaya, A., Krueger, I., Margaria, T., eds.: ICSOC 2008. (2008)
4. PMI: A Guide to the Project Management Body of Knowledge (PMBOK Guide). 4 edn. Project Management Institute (2008)
5. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM 26(11) (1983) 832–843
6. Reichert, M., Bauer, T., Fries, T., Dadam, P.: Realisierung flexibler, unternehmensweiter workflow-anwendungen mit adept. In Horster, P., ed.: Proc. Elektronische Geschftsprozesse–Grundlagen, Sicherheitsaspekte, Realisierungen, Anwendungen. (2001) 217–228
7. Zhou, J., Pakkala, D., Perala, J., Niemela, E.: Dependency-aware service oriented architecture and service composition. In: IEEE International Conference on Web Services. (2007) 1146–1149
8. Wu, Q., Pu, C., Sahai, A., Barga, R.: Categorization and optimization of synchronization dependencies in business processes. In: Proceedings of IEEE 23rd International Conference on Data Engineering (ICDE'07). (2007) 306–315
9. Zhou, Z., Bhiri, S., Hauswirth, M.: Control and Data Dependencies in Business Processes Based on Semantic Business Activities. In: Proceedings of iiWAS2008, ACM (2008)
10. Ensel, C., Keller, A.: An approach for managing service dependencies with xml and the resource description framework. Journal of Network and Systems Management 10 (2002) 147–170
11. Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M.C.: Monitoring Dependencies for SLAs: The MoDe4SLA Approach. In: IEEE SCC (1). (2008) 21–29
12. Karaenke, P., Micsik, A., Kirn, S.: Adaptive sla management along value chains for service individualization. In: Proceedings First International Symposium on Services Science (ISSS'2009). (2009)