Service Relationships Management for Maintenance and Evolution of Service Networks

Aneta Kabzeva, Joachim Götze, Thomas Lottermann, and Paul Müller
Integrated Communication Systems (ICSY), University of Kaiserslautern, Germany
Email: {kabzeva, j_goetze, t_lotterm09, pmueller}@informatik.uni-kl.de

***Abstract*—The Service-Oriented Architecture (SOA) paradigm is broadly accepted for the realization of business capabilities. Hence, the maintenance and evolution of Service Networks (SN) as systems comprising multiple service-based applications is becoming a growing issue. The larger a service inventory grows and the more often services are reused, the more consequences a service change or fault can cause on related applications in the SN. While reducing the adaptation complexity of a single solution, the realization of business processes as service compositions introduces logical relations defined implicitly between the technically independent services. To preserve the consistency in the whole SN, maintenance and evolution processes have to consider all relations to the changing configuration item. We present a framework for collection, validation, and representation of service relationship information. Contributions of the proposed solution include a semi-automatic approach for relationship identification, a mechanism for completeness and consistency validation, and a tailor-made representation of relations according to stakeholder needs.**

***Index Terms*—service-orientation; service networks; service relationships; maintenance; evolution**

I. Introduction

Nowadays, Service-Oriented Architecture (SOA) is the main paradigm applied for the flexible integration of heterogeneous applications. With the introduction of the core concept of a *service*, SOA aligns the development of business processes and the underlying IT infrastructure. From a business perspective, the decomposition of business processes into reusable services allows for easy recognition of relevant software components in case of changing product requirements and better overview of IT investments for the introduction of new business capabilities. For software architects, the realization of systems as service compositions means the definition of loosely coupled units of logic accessible through a standardized interface [8]. Thus, fast adaptation to changing business requirements is achieved through the modification or replacement of the service representing the relevant business task.

While the adoption of SOA reduces the complexity of single system adaptation, the structural complexity of a Service Network (SN) as a system of service systems [5] is increasing considerably. Therefore, *maintenance* (the modification of a service-based application for fault correcting or quality improvement changes of existing configuration items) and *evolution* (the introduction of new processes, services, and policies) are growing research issues [19]. Three main factors are identified for causing the increased complexity: the increased number of configuration items that can be considered for maintenance and evolution, the existence of implicitly defined dependencies, and the restricted administrative control on some resources within the landscape.

Increased number of configuration items: the decomposition of software solutions into a number of services and the definition of the expected documents describing their interfaces, compositions, and regulations increases the set of items which need change control [23].

Implicit dependencies across applications: the reusability of services in different processes speeds up new product implementations. Yet, it leads to an increasing number of relations between services in the context of these processes. Each service reuse generates hidden chains of dependencies [10]. These dependencies are not explicitly defined [16] and affect the maintenance and evolution of SNs.

Restricted administrative control: the standardized service access through uniform interfaces allows easy integration of third-party services. Such integration introduces configuration items which are under external administrative control and are needed for the proper functioning of applications. Changes conducted by the external providers cannot be controlled and can cause disruptions of client applications [27].

To exploit the agility provided by SOA, SN operators have to deal with the resulting complexity and assure consistent landscape state after maintenance and evolution changes. The loose coupling property of services provides only technical independence [26]. The modification of a service can still affect numerous processes and applications using the service. In the context of SNs, a change can cause not only functional but also non-functional faults such as the violation of contract clauses [33]. Proper propagation of an evolutionary or maintenance change through the entire Service Network requires a rigorous knowledge of the relationships resulting from service composition and reuse. A relationship between two entities can be a functional dependency or a non-functional requirement. The relationship management solution proposed here provides a means for collecting this knowledge, validating the completeness and consistency of the collected relationship information, and presenting it in a tailor-made form suitable for the analysis needs of both business and IT stakeholders. Based on predefined patterns and constraints, it calls attention to missing relationships and inconsistencies of specifications, yet leaves the correcting actions to human interaction. To achieve this goal, several steps have to be taken [15]:

- Understand the characteristics of services residing on the different abstraction layers between business and IT, as well as the possible relationships between them to provide a basis for a completeness and correctness check of the collected information.
- Define a common format for relationship representation allowing a uniform specification of all identified relationship variations, independent from the heterogeneous specification languages applied in the realization of service-based applications.
- Design an architecture capable of supporting the collection, validation, and representation of an appropriate set of relationships relevant for SNs according to their stakeholder needs.

This paper focuses on the last step and describes an architecture, a prototype, and an exemplary case study for a relationship management framework. The remainder of the paper is organized as follows: Section 2 gives an overview of currently existing solutions for relationships management for SOA. Section 3 identifies the relationship types considered for the proposed solution. Section 4 explains the proposed framework architecture. A prototypical realization is presented in Section 5. Section 6 describes the application of the framework in an exemplary case study. Finally, Section 7 concludes the paper with a short summary and an outlook on future work.

II. Related Work

The work on relationships in service-based applications found in the literature differs according to the purpose for relationship assessment and the considered set of relationships. Regarding the purpose for relationships, existing approaches separate in two general groups: providing support for business-specific purposes [3][24] or for IT-specific purposes [1][6][27][33]. The transfer of business requirements to the executable IT services [3] and automatic process model creation [24] are the main goals pursued from the business perspective. Regarding the type of relationships, these solutions are mainly interested in the mappings between business capabilities and the IT services responsible for their execution. The maintenance and evolution scenarios from the IT perspective include failure detection and impact analysis [1][2][12], definition of service level agreements for composed services [20][33], and governance support [6]. These solutions extract information mainly on the relationships between executable services. While providing some detail on relationships properties and analysis features for the specific scenario, all these approaches capture a restricted set of relationships types. The collected relationship information is not applicable for additional analysis purposes. The solution proposed in this paper considers these approaches as a basis to identify what types of relationships should be supported by the framework and what validation features are needed.

Infrastructures for generic traceability support are offered in [30][32]. Similar to our solution, the STraS framework [30] foresees plug-ins to extract data from heterogeneous specifications of architectural artifacts. However, the actual capturing of

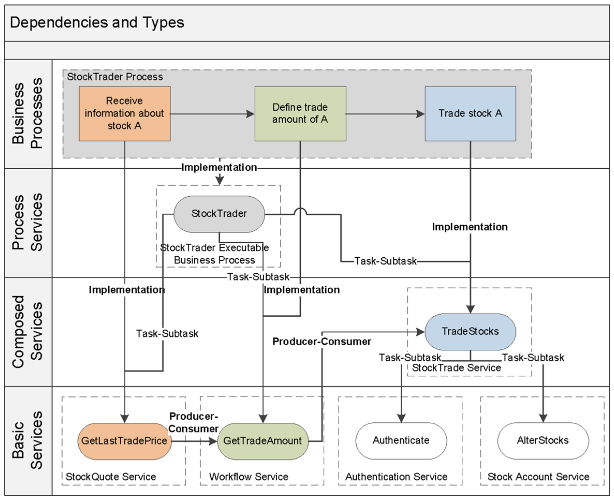

Fig. 1. Considered relationship types example

the relationships is not established. Stakeholders can query the ontology-based integrated knowledge representation according to their needs. Contrary to our approach, constraints and patterns for the validation of the collected relationships are not considered as part of the solution. The VbTrace approach [32] operates on an abstract and technology-specific process specifications. Based on a specified set of views, the View-based Modeling Framework (VbMF) produces links between view models and between elements from different view models instead of between SOA-specific architectural artifacts. From the captured relationships, the infrastructure allows code re-generation of the process implementation that should support changes in the process-driven SOA landscape. Although it considers the abstract business view on a process, this approach, and the restricted set of links that it generates, fits only a software developer's needs.

Realizing the role of knowledge management for the long-term success of service-based landscapes, several software vendor solutions for relationship management have emerged (e.g., IBM's WebSphere Service Registry and Repository [7], Software AG's SOA governance tool CentraCite [31], or Oracle's sCrawler SOA dependency tracker [28]). Typically, these solutions are built on top of a service registry and work only in combination with the corresponding vendor-specific software suite.

III. Service Relationships

The adaptability and flexibility of service-based applications is realized with the introduction of an additional *service abstraction layer* between business and application logic [8][14]. To structure the representation of both types of logic, the service layer is divided into several layers [8]. While there is a common understanding that the service layer comprises different types of services, there is no clear view what these types are. Service classification and how the different types relate to each other is normally dependent on the stakeholders' background [22]. However, independent from the number of abstraction layers defined for the realization of a SN, the aligned modification of the configuration items lying on these layers is what grants the success of a SOA and at the same time complicates its maintenance and evolution. Five maintenance and evolution challenges are identified according to the structure of SNs and addressed in the proposed solution: two address the impact estimation on business and IT change, two support the recognition of critical and wasted resources, and one regards the redundancy of services.

Business requirement change management: agile adaptation to changing requirements is the main reason for initiating a service-based application. A changing requirement has to be transferred to the execution services and will initiate changing processes on the IT side. Since business processes compose a set of business tasks, a change request for a single business task will directly trigger a change of a restricted set of services responsible for its execution. Yet, the proper functioning of services executing adjacent business tasks - which can also be part of multiple business processes composing the modified task - can also be indirectly affected.

Service change management: service quality improvement or fault correction are possible triggers for service changes coming from the IT side. Even if the service interface remains the same, a service change can have implications on the non-functional properties of the supported business capabilities. Because of the reuse of services, there can be an n to m relationship between services and tasks [30]. Multiple business tasks, and consequently business processes, can be affected.

Detection of critical services: the maintenance process is not only responsible for the execution of modifications on change requests, but also for ensuring a high quality application environment. A service is provided for consumption to an undefined set of consumers. Its usage after deployment is unpredictable for the provider. Without information on the connectivity of a service within the entire SN, it is impossible to recognize which services are crucial for the business.

Detection of wasted resources: similar to the previous challenge, service consumption is also impossible to estimate for unused services without keeping information on their connectivity. An unused service wastes storage resources or even monetary resources in the case of a third-party service. Integrating data from usage accounting can provide information about the significance of the resource waste [11].

Service redundancy prevention: a service is redundant if there is already another service offering the same functionality with the same quality for the same business capability. Services and processes are procured or provided by different stakeholders shaping the SN. A system architect cannot know all available services unless there is an explicit documentation on how the available services relate to business tasks.

To support these challenges, the proposed solution allows the extraction and explicit documentation of the relationships described below. Fig. 1 provides an example illustrating their occurrences. This figure depicts a manually created relationship model of an existing SOA-based stock trading application, which will later be considered as an exemplary case study.

Task-to-task: This relationship type represents links within the business process layer. A relationship between tasks displays the control flow (*control relationship*) or data flow (*producer-consumer relationship*) within an application. This information is explicitly available in business process descriptions and automatically extractable for relationships within a single process. A complete task-to-task relationship model for a task requires reviewing all the processes comprising the task, which is a time consuming activity without an automated relationship management solution. Task-to-task relationship information can be used to automatically map relationships between services on the executable services layers, which result in the process context and are not explicitly visible for software architects. Thus, support for service change management and the estimation of the connectivity of a service within the SN will be indirectly provided.

Task-to-service-operation: To be reusable, an executable service is usually entity-centric, defining a set of operations on a single business entity. A task within a business process defines a piece of functionality. Thus, a business task is usually executed by a single or multiple operations provided by one or more services. Providing explicit information on task-to-service operation mappings supports both business requirements change management and service change management. Because of the fine granularity of the relationship not only to a service but to its specific operation, responsible stakeholders will be able to better estimate if all related services or tasks will be affected by a change request. Furthermore, for the definition of new business processes composing an existing service task, the corresponding service can be automatically detected and prevent unwanted service redundancies. Yet, the collection of these relationships has to happen manually on the initial service selection from the software architect.

Service-operation-to-service-operation: Links within and across the executable services layers are displayed by these relationships. Service-operation-to-service-operation relationships can be captured in two ways. Relationships that are automatically collected from process services or composed service specifications residing on higher abstraction layers indicate functional dependencies (*task-subtask relationships*) across layers. Relationships that are automatically calculated from the combination on the previous two types of relationships inherit the type of the initial task-to-task relation (e.g., *producer-consumer* in Fig. 1). Both types support traceability for change management and representation of the service integration within the SN.

Business-process-to-service: Finally, if the logic modeled within a business process is controlled by an executable process service, a process-to-service implementation relation has to be explicitly captured for change management support.

IV. Service Relationship Management Framework

A. Requirements

To provide a framework for relationship management in SNs, several requirements have to be taken into account. First of all, the set of configuration items and their relationships have to be *dynamically adjustable to the target landscape*. Depending on the maturity of the SOA adoption within an organization [9], only a subset of the layers pictured in Fig. 1 may be present. A restricted set of service layers will result in a smaller set of possible relationships. The framework should not require a specific set of configuration items and relationships to be available, but adjust to the available infrastructure and its rules and constraints. Thus, an organization can adjust the completeness and correctness validations performed according to its infrastructural policies.

This leads to the second requirement: the relationship management solution should be *applicable for both existing and newly forming SNs*. To achieve this requirement, the solution should operate on available documentation without the need for modification.

Another requirement on the framework is to *capture both direct and indirect (hidden) relationships* between the configuration items of a SN. While direct dependencies can be easily extracted, either manually during design or automatically from process descriptions indirect dependencies, resulting from hierarchical service compositions and reuse in multiple processes, can be discovered only by tracing multiple descriptions, originating at different times from different stakeholders.

In highly mature SNs with repeated and augmented service compositions [9], the collection of all existing relationships can be a long running process. The more often a service participates in compositions, the higher the number of its relationships to other services will be. The more compositions in a service-based landscape exist, the higher the probability of hidden relationships. Therefore, *once calculated, relationships models should be cached and re-evaluated only on modifications*. This grants both the freshness of the model and better performance.

The relationship models acquired with the framework should be usable for different maintenance and evolution issues like change management or architecture quality analysis. Change analysis can be triggered from a modification request on a single service. A relationship model of interest, in this case, should visualize all configuration items within the infrastructure that could be influenced by the service modification. For an architecture quality analysis, the software architect can be interested in the topology of the whole infrastructure to identify business-critical services. Depending on the purpose of a stakeholder, the content of a relationship model view will differ. A *view-based representation* of the collected relationship information should be prepared from the framework to improve the usability of the models for different stakeholder groups.

B. Architecture

The architecture proposed here for relationship management in SNs comprises three horizontal layers (see Fig. 2): a *relationship collector* layer responsible for extracting relationship information from configuration item descriptions, a *relationship profiler*, which calculates additional relationships based on multiple inputs from the relationship collector, and a *relationship presenter* layer, which prepares the relationship information for stakeholder-specific extraction and visualization. A vertical layer, *relationship constraints and patterns*, supports the three horizontal layer activities through the definition of patterns and constraints specific for the structure of service-based application landscapes. The whole architecture is positioned on top of the service-based application infrastructure to be captured. It works on the basis of existing items' descriptions without requiring any specific language or additional tagging in the specifications, thus addressing the first two requirements on the desired solution.

The *collector layer* processes raw data from configuration item descriptions and transforms it into an uniform specification of the configuration item and its direct relationships according to the item-specific relationship patterns provided from the vertical layer. To support an extensible set of configuration item types, the collector layer has an extensible, modular structure. For each type of configuration item, a specialized collector module is provided that knows what type of information to search its documentation for. All recorded dependencies are presented as first-class entities in the uniform specification format and are passed for further processing to the relationships profiler layer. To be able to understand different modeling notations, like BPMN (Business Process Model and Notation) [25] or EPC (Event-driven Process Chain) [29] for business process descriptions, a set of patterns mapping the notation-specific structures to the unified information model should be provided to the collector.

The objectives of the *profiler layer* are the calculation of indirect relationships and the validation for completeness and inconsistencies. While a relationship collector processes one item description at a time, a relationship profiler combines the information from multiple descriptions. Compared to the relationship collectors, which have to be language-specific, a relationship profiler works on a unified set of data and is thus language-independent. Again, following the extensible approach advanced by the previous layer, an item-specific profiler determines how to search for relevant description files for a calculation. The completeness and consistency checks are done against item-specific constraints, defined in the vertical architectural layer on the basis of possible relationships. The completeness of the collected data is dependent on the content provided in the underlying item specifications, e.g., all services invoked in a composed service are captured as part of the landscape model. To allow for completion of mandatory information, the collection layer has to notify the responsible stakeholder and request for missing inputs, e.g., initial task-to-service operation record. In case of inconsistency, the framework only notifies the responsible stakeholder. The goal of the framework is to capture the structure of an application landscape and not its correction, which is in itself a complex issue usually requiring human interaction. To map the landscape architecture as it is, inconsistent relationships have to be kept within the model until their correction through the modification of the related configuration items. The modification will trigger a re-calculation of the

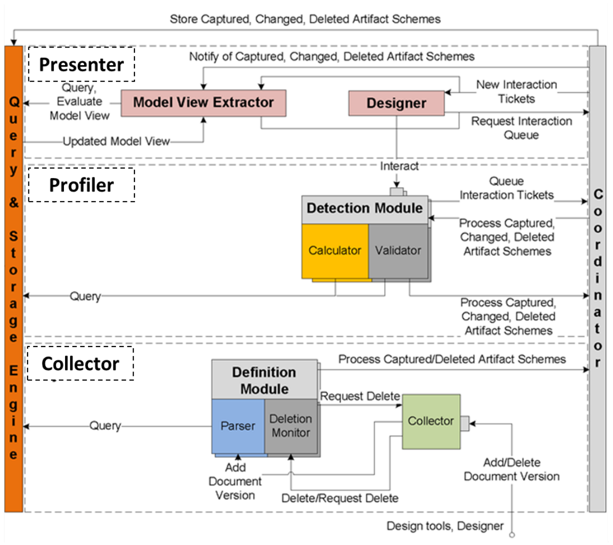
Store Captured, Changed, Deleted Artifact Schemes
Presenter
Notify of Captured, Changed, Deleted Artifact Schemes
Query, Evaluate Model View
Model View Extractor
Designer
New Interaction Tickets
Request Interaction Queue
Updated Model View
Query & Storage Engine
Interact
Profiler
Queue Interaction Tickets
Detection Module
Calculator
Validator
Process Captured, Changed, Deleted Artifact Schemes
Query
Process Captured, Changed, Deleted Artifact Schemes
Coordinator
Collector
Definition Module
Process Captured/Deleted Artifact Schemes
Request Delete
Parser
Deletion Monitor
Collector
Query
Add Document Version
Delete/Request Delete
Add/Delete Document Version
Design tools, Designer

Fig. 2. Architecture of the relationship management framework

relationships in the profiler, which will update the relationship model. The calculated relationship profiles are finally saved for stakeholder-based representation and reasoning.

The topmost *presenter layer* handles the preparation of the calculated relationship models for representation, according to the analysis-specific content requirements of a stakeholder. An automatic selection of the desired subset of relationships and configuration items needs predefined rules. Based on their role [9] in the operation of the service-based landscape, stakeholders can choose what part of a calculated relationship profile should be shown. Thereby, they should be able to restrict the visibility of configuration items as well as the type of relationships between them. The extraction of model views reduces the complexity of the model and improves its readability for stakeholders by presenting information according to their domain expertise.

V. Realization

The prototypical implementation of the framework uses the Service Component Architecture (SCA) programming model with its Apache Tuscany implementation [18] to support the development of a flexible service-based framework. The distribution of the prototype components on the three horizontal abstraction layers is depicted in Fig. 2.

The collector layer comprises a *Collector* and a set of *Definition Modules*. The collector provides an entry point for new configuration item descriptions to the framework. It acts as a central definition hub, which forwards provided definitions or configuration items deletion requests to the responsible definition modules, based on the description's type. The definition modules have the task to parse definitions of configuration items and monitor the deletion of already collected ones. The *Parser* within a definition module translates the language-specific description of a configuration item into a generic data structure which is used within the framework and marks it with a unique ID for the landscape. It also extracts notation-specific dependencies when available (e.g., task-to-task dependencies). The prototype provides definition modules for WSDL (Web Services Description Language) [4], BPEL (Business Process Execution Language) [21], BPMN, and EPC. The *Deletion Monitor* is polled every time an artifact shall be removed. When a deletion request arrives, the monitor either grants the request or throws an exception, depending on the relationship information found in the landscape model. Since only the collector is known to external design tools, it is possible to transparently integrate new definition modules for new types of configuration items for the stakeholder. No new skills or client adaptations are required in order to use the framework.

Newly captured configuration items, now represented in the generic data structure, are forwarded to the *Coordinator*. The coordinator is the central controlling unit. It forwards the configuration items to relevant detection modules for relationship profile calculation and validation. Then it sends the new information (relationships and configuration items) to the query and storage engine, and notifies the presentation components about the changes.

The functionality of the profiler layer is implemented as a set of *Detection Modules* responsible for calculating implicit relationships. Each detection module consists of a *Calculator* and a *Validator*. A calculator implements a detection algorithm for an implicit relationship type. The validation is performed in terms of completeness (whenever a component or information is missing which is needed to extract mandatory information) and consistency (whenever a potential problem embedded within the application landscape is discovered based on contradicting relationships). Each validation issue generates a ticket with a priority tag to designate the importance of its processing. The current prototype gives higher priority to consistency issues. To implement the collection of the relationships specified in section three, the prototype provides three detection modules: a mapping detection module, which collects and validates task-to-service operation and process-to-service relationships, an inter-service dependency detection module, which calculates and validates service-operation-to-service-operation relationships, and a service classification detection module capable of classifying and validating an executable service automatically. Through a concept of pluggable detection modules, the insertion of new relationship types is achieved by simply binding a new module to the coordinator.

The *Query and Storage Engine* provides an interface to the data storage containing the captured relationship models. The prototype saves the data in a DOM tree, which is managed via JDOM [13], allowing XPath processing.

The presenter layer consists of two types of clients: the *Model View Extractor*, which provides a graphical representation of the collected relationships model (cf. Fig. 3) and the *Designer*, which allows the stakeholders to contribute to the collection process. The model view extractor tool allows stakeholders to create model views based on XPath expressions and displays only a specific part of the model as a graph. The generated graphs are automatically updated whenever their

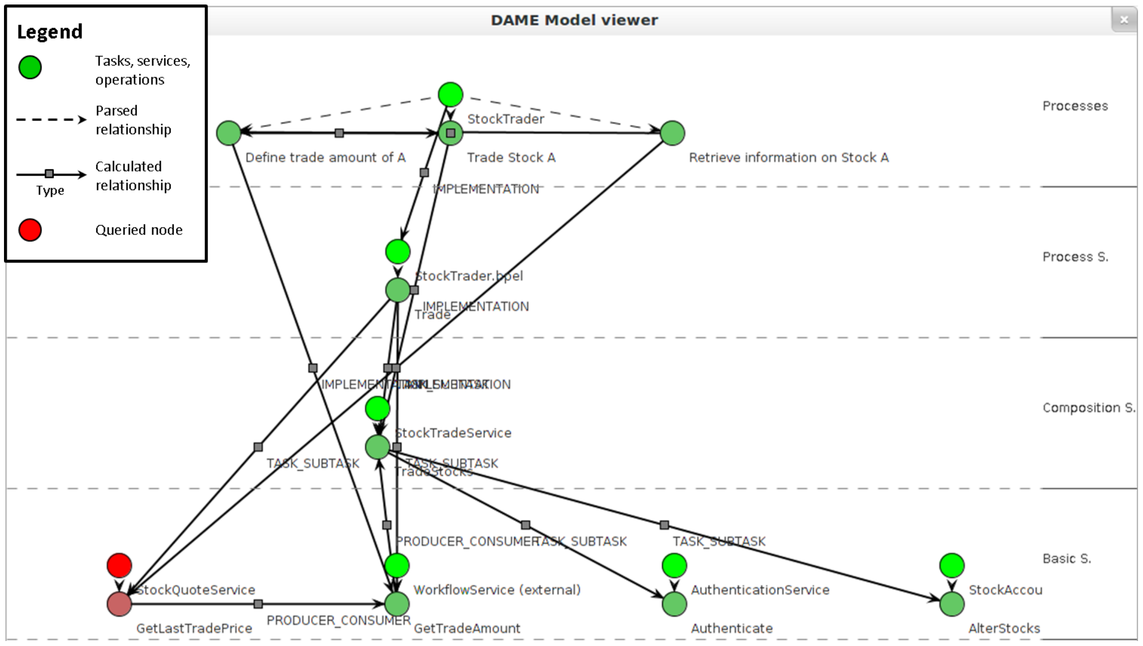

Fig. 3. Automatically generated relationships model of the stock trade application

content is affected by modifications in the architecture. The designer is a tool for manual interaction with the framework for inserting, changing, and deleting configuration items in the application landscape. In addition, it interacts with the detection modules in order to solve the validation issues recognized by the framework. Thus, the designer tool allows the framework to extract dependencies which require manual interaction as well as solve potential problems discovered within the landscape during validation.

VI. Exemplary Case Study

The proposed framework was applied to collect, validate and represent the relationships in the service network case from Fig. 1. The prototype was used to assess and validate the structure of the SN during an exemplary creation of the stock trader application. The goal was to observe the framework's behavior under conditions like missing or incomplete documentation and false service classification. For this purpose, application creation was simulated with the following steps:

1) A business analyst defines a BPMN business process description for the *StockTrader Process*.
2) A software developer defines three WSDL descriptions of the services responsible for the implementation of the process specified in step 1 - *StockQuote Service*, *Workflow Service*, and *StockTrade Service*.
3) A software architect defines the executable BPEL process specification for the *StockTrader Process*.
4) A BPEL specification of the *StockTrade Service* is imported in the network.
5) The service descriptions for the *Authentication Service* and the *StockAccount Service* composed by the *StockTrade Service* are added to the network.

The result from the first step was a business process comprising three tasks with two explicit task-to-task relationships between them. Additionally, for each of the three tasks a notification concerning the missing implementation of the tasks discovered within the process description was generated.

After the second step three basic services were added to the model with no relations. Three additional notifications of unused services were received. The stakeholder was advised to free unnecessarily used resources or provide, via the Designer tool, the mapping of which business task is implemented by which service operation. The notifications were addressed by providing the implementation relations between the three tasks and services via the Designer.

The analysis of the BPEL description from step 3 resulted in adding a process service with three task-subtask and two producer-consumer relations to the model graph.

The re-validation of the model in step 4 after inserting a BPEL description for the basic *StockTrade Service* led to a classification inconsistency. Also, two unknown service descriptions referenced in the BPEL specification were reported and asked for their insertion. The analysis of the BPEL specification resulted in the automatic relocation of the *StockTrade Service* to the composition service layer.

Addressing the requests for the service descriptions for the *Authentication Service* and the *StockAccount Service* in step 5 generated the two task-subtask relations from the *StockTrade Service* to its composite services. The resulting relations model (see Fig. 3) was automatically drawn by the framework and represents all relations from the manual assessment in Fig. 1. It shows the connectivity of the *StockQuote Service* (colored in red) within the network.

VII. Conclusion and Future Work

Understanding and explicitly modeling relationships in SNs is an essential prerequisite for controlled maintenance and evolution. This paper proposed an architecture for capturing and validating explicit and implicit service relationships. The approach considers that the different configuration items in a Service Network are specified in existing heterogeneous description languages and applies a language-independent relationship specification model to store the connectivity within the landscape. Implicitly defined dependencies resulting from service composition and reuse are captured in an explicit way, providing information on the relationship type. Applying predefined rules for relationship obligation and consistency violation, the proposed solution considers validation of the captured landscape model for completeness and consistency. For every validation issue, tickets for stakeholder interaction are generated and motivate the enhancement of the landscape infrastructure. Finally, respecting multiple stakeholder roles from the business and IT domain in a SN, and their different analysis needs on the service-oriented infrastructure at place, a view-based representation of the captured information has been considered as part of the presented framework. The application of our solution in an exemplary case study providing typical descriptions for service-based applications shows that all relationships identified as helpful for both business analysts and software architects for the decision making process during change management are captured automatically by the framework by complete landscape documentation. Incomplete documentation is discovered by the framework and reported to the relevant stakeholders.

Next steps to further improve the relationship management approach include testing of the framework capabilities and extending the validation range. Evaluations against the SAP R/3 [17] processes should assess the behavior of the prototype in a more complex service-based landscape with hundreds of processes and services. The EPC definition module necessary for this purpose is already implemented and integrated within the prototype. To increase the validation range, an exhaustive set of relationship patterns and constraints based on the architectural peculiarities of service-based infrastructures will be elaborated and integrated in the solution.

References

[1] S. Basu, F. Casati, and F. Daniel, ”Toward Web Service Dependency Discovery for SOA Management,” IEEE International Conference on Services Computing (SCC 2008), Honolulu, 2008, pp. 422-428.

[2] L. Bodenstaff, A. Wombacher, M. Reichert, and R. Wieringa, ”MaDe4IC: An Abstract Method for Managing Model Dependencies in Inter-Organizational Cooperations,” Service Oriented Computing and Applications, vol. 4, no. 3, 2010, pp. 203-228.

[3] S. Buchwald, T. Bauer, and M. Reichert, ”Bridging the Gap Between Business Process Models and Service Composition Specifications,” Service Life Cycle Tools and Technologies: Methods, Trends and Advances, 2011, pp. 124-153.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, Web services description language (WSDL) 1.1, W3C submission, 2001.

[5] O. Danylevych, D. Karastoyanova, and F. Leymann, ”Service networks modelling: An SOA & BPM standpoint,” Journal of Universal Computer Science, 2010, pp. 1668-1693.

[6] P. Derler and R. Weinreich, ”Models and tools for SOA governance,” Lecture Notes in Computer Science, vol.4473. Springer Verlag, Berlin, Heidelberg, 2007, pp. 112-126.

[7] C. Dudley, L. Rieu, M. Smithson, T. Verma, and B. Braswell, WebSphere Service Registry and Repository Handbook, IBM Redbooks, 2007.

[8] T. Erl, Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall, 2005.

[9] T. Erl, S.G. Bennett, C. Gee, R. Laid, A.T. Manes, R. Schneider, L. Shuster, A. Tost, and C. Venable, SOA Governance, Prentice Hall, 2011.

[10] S. Frischbier, A. Buchmann,and D. Pütz, ”FIT for SOA? Introducing the F.I.T.-Metric to Optimize the Availability of Service Oriented Architectures,” Second International Conference on Complex Systems Design and Management (CSDM 2011), Paris, France, 2011, pp. 93-104.

[11] J. Götze, T. Fleuren, B. Reuther, and P. Müller, ”Extensible and scalable usage accounting in service-oriented infrastructures based on a generic usage record format,” 6th International Workshop on Enhanced Web Service Technologies, ACM, 2011, pp. 16-24.

[12] M.A. Hirzalla, A. Zisman, and J. Cleland-Huang, ”Using Traceability to Support SOA Impact Analysis,” IEEE World Congress on Services, Washington, DC, 2011, pp. 145-152.

[13] J. Hunter and B. McLaughlin, JDOM, http://jdom.org/, 2012.

[14] N.M. Josuttis, SOA in Practice, O'Reilly, 2007.

[15] A. Kabzeva and P. Müller, ”Toward Generic Dependency Management for Evolution Support of Inter-Domain Service-Oriented Applications,” European Conference on Service-Oriented and Cloud Computing (ESOCC 2012) PhD Symposium, Bertinoro, Italy, 2012, pp.35-40.

[16] A. Keller and G. Kar, ”Determining service dependencies in distributed systems,” IEEE International Conference on Communications (ICC 2001), Helsinki, Finland, 2001, pp. 2084-2088.

[17] G. Keller and T. Teufel, SAP R/3 Process Oriented Implementation, Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1998.

[18] S. Laws, M. Combellack, R. Feng, and H. Mahbod, Tuscany SCA in Action, Manning Publications Co., 2011.

[19] G.A. Lewis and D.B. Smith, ”Service-oriented architecture and its implications for software maintenance and evolution,” Frontiers of Software Maintenance (FoSM 2008), Washington, DC, 2008, pp. 1-10.

[20] A. Ludwig and B. Franczyk, ”COSMAAn Approach for Managing SLAs in Composite Services,” ICSOC 2008, Springer-Verlag Berlin Heidelberg, 2008 (LNCS 5364), pp. 626-632.

[21] OASIS, Web Services Business Process Execution Language Version 2.0. OASIS Standard, 2007.

[22] P. Offermann, C. Schröpfer, O. Holschke, and M. Schönherr, ”SOA: The IT-Architecture behind Service-Orientation,” Workshop MDD, SOA and IT-Management, Oldenburg, Germany, 2007, pp. 1-11.

[23] Office of Governance Commerce (OGC), ITIL v3: Information Technology Infrastructure Library Version 3, volume 1-5. London: The Stationary Office, 2007.

[24] A.M. Omer and A. Schill, ”A Framework for Dependency Based Automatic Service Composition,” Business Process Management Workshops (BPM 2008), Milano, Italy, 2008, pp. 535-541.

[25] OMG, Business Process Model and Notation (BPMN) Version 2.0, OMG Specification, 2011.

[26] M.P. Papazoglou, ”Service-Oriented Computing: Concepts, Characteristics and Directions,” 4th International Conference on Web Information Systems Engineering (WISE 2003), Rome, Italy, 2003, pp. 3-12.

[27] L. Pasquale, J. Laredo, H. Ludwig, K. Bhattacharya, and B. Wassermann, ”Distributed cross-domain configuration management,” Service-Oriented Computing, Springer, pp. 622-636.

[28] S. Phukan, sCrawler: SOA Dependency Tracker, Oracle Technology Network, 2009.

[29] A.W. Scheer, O. Thomas, and O. Adam, ”Process modeling using event-driven process chains,” Process-Aware Information Systems: Bridging People and Software through Process Technology, 2005, pp. 119-145.

[30] S. Seedorf, K. Nordheimer, and S. Krug, ”STraS: A Framework for Semantic Traceability in Enterprise-wide SOA Life-cycle Management,” 13th Enterprise Distributed Object Computing Conference Workshops, 2009, pp. 212-219.

[31] Software AG, CentraSite Governance Edition, User's Guide 7.1, 2011.

[32] H. Tran, U. Zdun, and S. Dustdar, ”VbTrace: Using View-based and Model-driven Development to Support Traceability in Process-driven SOAs,” Software and System Modeling, vol. 10, no. 1, 2009, pp. 529.

[33] M. Winkler, T. Springer, E.D. Trigos, and A. Schill, ”Analysing dependencies in service compositions,” Service-Oriented Computing ICSOC/ServiceWave 2009 Workshops, Springer, pp. 123-133.