Relaxed Service-type matching and Transformation management

Lea Kutvonen

Department of Computer Science, University of Helsinki, Finland
Lea.Kutvonen@cs.Helsinki.FI

1 Introduction

Current technological developments with service oriented architectures (SOA) [1], model-driven techniques − with model repositories (e.g., ebXML [2]) and heterogeneous implementation frames (e.g., MDA [3]}) − and middleware level interoperability facilitate inter-enterprise collaboration. Thus the development of loosely-coupled cross-organisational business networks is becoming more mature.

The web-Pilarcos and Pilarcos projects [4,5] have worked towards an experimental *interoperability middleware*. The web-Pilarcos B2B middleware provides services for managing the life-cycle of dynamic business networks in an inter-enterprise environment. The middleware supports concepts of eCommunity, eCommunity contract, business policies, and operational time conformance to the contracted business network model. The middleware services aim for a rigorous level of transparent interoperability support as part of the eCommunity life cycle services. Aspects of interoperability include awareness of collaboration processes, and collaboration level adaptation to breaches in operation.

In this environment, there are three interoperability challenges of interest. First, the services provided for the business network should match the purpose and role division of the network. Here the semantics of the services is of importance, especially the information flow between role holders in the network. Second, the service interfaces provided by the actual implementations should work together, although they do not need to be identical. There are lots of software engineering methods for designing, implementing and generating interceptors / transformers to plug service users and service providers together (e.g., connectors [6]). However, most of them consider integration or configuration of modules into a larger composition of objects or components. In the inter-enterprise environment we are more concerned of finding services that are interoperable in terms of the provided and used service interfaces, and furthermore, allow some relaxation in the matching process. Finally, below the application level semantics of the interactions, there is a layer of communication services: transfer of messages, encoding of information, and layers of support protocols for achieving transactions, secure and private communication, QoS management, etc.

This paper discusses solutions to the second challenge. During the business network establishment phase, the web-Pilarcos middleware tries to ensure that interacting services have interoperable interfaces. This is done using the help of federated type repositories, where service type descriptions are stored and together with some verified relationships between known service types. As an important property, each relationship carries a reference to the interceptor that needs to be placed between those interfaces to gain interoperability.

2 Relaxed service-type matching

Although much of service discovery research, like UDDI and semantic web extensions, is directed towards user-oriented browsing and discovery of human-usable services, we aim to software-composition oriented type matching for interoperability purposes. In this case, the directive ontology is derived from the three challenges noted in above. First, the business network requirements for a service are to be derived from the role requirements. Second, the implementor needs to express the interface description of the service provided for matching and discovery purposes. This is further detailed below. Finally, the service description itself does not involve the abstract communication layer concerns, but those are addressed separately by a compulsory part in the service offers collected to a service offer repository. The service matching process has to check both parts, but separately: as a result, interceptors can be added for both levels independently [7].

The service descriptions are based on predefined service types. All parties of the common network are allowed to define new service types, so that an evolving type system is created. A *service type* defines functional and non-functional properties for a class of business services. The functional part of a service type comprises of interface signature (service interface syntax), interface protocol and additionally semantic annotations for exchanged documents (messages). The interface protocol describes the externally visible behaviour of a service in a bilateral conversation. The non-functional properties of a service type describe for example QoS-requirements and policies. When a new service is published to a public service offer repository its behavioural properties and especially its conformance to the claimed service type must be verified. Behavioural descriptions of service types are also needed for static verification of service interoperability and runtime monitoring of conformance between the community contract and actual service behaviours.

Often, only equality or subtyping relationships are considered when service types are matched. Or, as with semantic web, there is an ontology into which services are grouped and matching descriptions can be found based on hierarchical positioning.

Here, we form the relevant service type ontology little by little into the type repository system. The ontology is fairly flat, but wide: as the users of the repository are middleware agents, they cannot adapt to generalizations or specializations of a service type very far, but are more agile in plugging new technology dependent pieces into an already existing framework. The main purpose of the type repository is to allow checking that service types match together, and to give references to small modules needed in the framework to patch minor technical differences. Therefore, there is an technology independent level of services that is concerned with information exchange relationships, and a more technology oriented level that is concerned on information representation and application level protocols. This is adequate, taken that a further technology dependent layer is separately organised to support these selections.

The relationships of interest for the type repository users are: no match, similar types (equality of text or reference, subtyping), and interoperable with interception. The comparison and judgment is not fully automated and cannot be made (due performance issues) at the time of query. Instead, the service type publication process involves verification of the type, comparison to other named types, and verification of the type relationships. The process of interceptor creation is external to the type repository.

3 Type repository services

The initial Pilarcos type repository was developed during the work on the ODP type repository function standard [8] and OMG MOF specification [9]. Although there are certain differences, most interfaces are similar. Thus the type repository offers operations [7] for

- publishing realizations of abstract types,
- checking whether two type realizations are conformant and interchangeable,
- retrieving subtypes or supertypes of a type realization,
- retrieving templates for a given abstract type,
- translating one type realization to another,
- retrieving names for abstract types, and
- type realizations in other type domains.

The type repository information base is organized by abstract services types; for each of them there is a set of concrete service descriptions. In contrast to the original type repository, where templates were stored into the common repository as well, we have diverged in web-Pilarcos, and take all technology and enterprise specific templates to be stored locally.

Another development step is in the service interface descriptions, where we have moved from the OMG IDL descriptions to Web Services oriented WSDL descriptions, enhanced with annotations for nonfunctional aspects.

The service type descriptions stored have to reflect two important design issues: federation of type repositories for the global network, and access performance. For these, we allow relationships to be defined across individual type repositories. In addition, a cache system and appropriate data partitioning are required to improve performance.

Technically, the type repository items [7] consist of

- type repository interface reference,
- type name to be interpreted via that interface,
- classification for the abstract service type (service type, interface type, behaviour type, etc. to help the type repository server to choose a structure for interpretation),
- names of component types (such as signature and behaviour type names),
- immutability policy (such as `immutable', `mutable using protocol', `temporary'),
- notification protocol for modified definitions,
- relationships to other types either within the same type domain or in other domains (such as `equal', `subtype', `interceptable'),
- interceptor interface reference, and
- cache for type descriptors from other type repositories, time-to-live for cache, and re-evaluation instructions.

Currently, we are enhancing the type system to take into account protocol-based behaviour [10,11].

Effectively, the type repositories form a distributed naming system for service types that are described in more or less heterogeneous style [12]. The pragmatical needs for finding relationships between definitions drive type publishers to create interceptors, and to verify relationships so that they are accepted and stored to the type repository.

4 Conclusions

The federated type repository service is an essential element of a B2B middleware that supports establishment of new business networks, or in a more simple case, connection between independently administered clients and servers.

The role of the type repository is to provide a trustworthy source of service type information, and furthermore, provide transformation services for communication between almost similar interfaces.

The service types can thus be matched with each other in a more relaxed way, only limited with interoperability requirement. As an enhancement, the cost of connection can be added to direct users to choose "native" types instead of transformed connections.

The service type matching approach supports evolution of services in a heterogeneous environment, where independent actors create new items, and where market forces has effect on the usability of items, in addition to the verifiable correctness properties. Furthermore, the approach gives a natural tool for managing one type of transformation components needed in the current component-based, model-driven networking environment.

Bibliography

1. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing. Communications of the ACM 46:10 (2003). pp. 24--28.
2. ebXML Technical Architecture Project Team: ebXML Technical Architecture Specification v1.0.4. Technical report, ebXML (2001) http://www.ebxml.org/specs/ebTA.pdf.
3. Siegel, J.: Developing in OMG's Model-Driven Architecture. Object Management Group. (2001) White paper, rev. 2.6.
4. Kutvonen, L., Ruokolainen, T., Metso, J., Haataja, J.: Interoperability middleware for federated enterprise applications in web-Pilarcos. In: First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05).
5. Kutvonen, L.: Automated management of interorganisational applications. In: Enterprise Distributed Object Computing (EDOC2002).
6. Balek, D.: Connectors in Software Architectures. PhD Thesis, Charles University, Czech Republic (2002).
7. Kutvonen, L.: Trading services in open distributed environments. PhD thesis, Department of Computer Science, University of Helsinki. (1998)
8. ISO/IEC JTC1: Information Technology~--~Open Systems Interconnection, Data Management and Open Distributed Processing. Reference Model of Open Distributed Processing. ODP Type repository function}. (1999) IS14746.
9. Object Management Group: Meta-Object Facility (MOF).
10. Ruokolainen, T.: Component interoperability. Master's thesis, University of Helsinki, Department of Computer Science. (2004) In Finnish.
11. Ruokolainen, T.: Type management for service oriented computing. In: First European Young Researchers Workshop on Service Oriented Computing. (2005)
12. Kutvonen, L.: Challenges for ODP-based infrastructure for managing dynamic B2B networks. In Vallecillo, A., Linington, P., Wood, B., eds.: Workshop on ODP for Enterprise Computing (WODPEC 2004). pp. 57--64.