A Component System with Automatic Service Composition for Spontaneous Virtual Networks

Philipp Schaber
Department of Computer Science IV
University of Mannheim
Mannheim, Germany
schaber@informatik.uni-mannheim.de

Wolfgang Effelsberg
Department of Computer Science IV
University of Mannheim
Mannheim, Germany
effelsberg@informatik.uni-mannheim.de

***Abstract*—We propose to demonstrate a component system and framework in the context of overlay networks, which can be used to dynamically create and set up modularized network services. A composer can automatically compose such services from a library of available components, taking into account dependencies among components and the possibility of sharing components by different services. In addition to group communication and publish/subscribe functionalities, which are implemented as components already, it is very easy to extend the library with third-party components or own implementations. The framework takes care of the calculation of a suitable service composition, and its deployment among all participating nodes. As the components are linked at runtime, each composition can take into account varying application requirements or incorporate new components without changing the host application. Since the system is also highly modularized itself, is allows to flexibly choose alternative composition or deployment algorithms, or even develop new ones.**

I. Motivation

Many different kinds of network overlay techniques and implementations exist today, allowing to create virtual overlay networks on top of an existing physical network. However, for an application developer, this also implies several drawbacks: First, each implementation will most likely have its own interfaces and dependencies. This renders it very difficult to try or switch different techniques and overlays – even just for comparison purposes. Besides different usage and interfaces, you manually have to install and link libraries it depends on. Also, as long as you do not compile all alternatives into your application, you cannot dynamically select the one that is fitting best (e.g., depending on the underlying kind of network). Third, if you use multiple overlay services, you might create redundant and unnecessary traffic because each service creates it's own control overlays. For example, although a single DHT could be shared, each overlay service will create and use its own one. Fixing this for monolithic implementations would require fundamental changes to the used libraries.

We therefore propose a system to modularize network components, especially for the field of virtual networks using overlay services. Our framework provides a component management, automatic composition, and deployment techniques for dynamically created overlays. It is modularized itself to provide a high flexibility.

II. Scope and Related Work

To set the scope of our work and differentiate it from similar approaches, the basic principles, related fields, and relevant differences are briefly described.

Our approach is to have a underlying component system as basis, with components that are not required to adhere to *one* specific, network-related interface. Instead, different functionality can be defined each by its own interface, with components being able to then offer a certain functionality by implementing the according interface. Although the component framework is thus not strictly bound to networking in principal, it is targeted mainly at creating overlay services and additionally provides the means to deploy a service among all participating nodes in a network.

This is in contrast to the field of *protocol composition*, which usually has stricter specifications on protocol components, with predefined channels for communication, as they are usually tightly integrated into a regulated protocol framework. On the other side, *service overlay networks* (SONs) and *pervasive computing* use similar concepts on interfaces and composition, but target another goal: They use overlays to compose arbitrary pervasive/web services, where different nodes provide different parts of the final service. On the contrary, our target is to support the modular creation of overlay-based services for a distributed service among *all* nodes. Different components and overlays will be used and combined to form a service, but this composition is then equal on all participating nodes.

III. SpoVNet – Spontaneous Virtual Networks

SpoVNet [1] is a research project that develops overlay-based tools and techniques for easy development and spontaneous deployment of distributed network applications and services.[1] As part of the SpoVNet project, ariba [3] is developed at the Karlsruhe Institute of Technology (KIT)[2]. It is the base for all other SpoVNet software and provides a self-organizing transport connectivity across different heterogeneous networks with an ID-based addressing. Its two main conceptual solutions are: First, the *Base Overlay*, which brings together all

[1] http://www.spovnet.de
[2] http://www.ariba-underlay.org

participating nodes in a unified communication context and also provides bootstrapping mechanisms for a virtual network. A decentralized KBR overlay is used as routing and control structure. Second, the *Base Communication,* which provides end-to-end connectivity in face of heterogeneous networks. It has a self-organizing detection of relay paths and relay responsibilities.

Our framework uses ariba, which is encapsulated in a wrapper component, as base communication for certain core functionalities (such as the composition deployment). Additionally, any component can have a dependency on ariba to use its functionality. However, the framework is not depending on SpoVNet in principal. As the core functionality is modularized itself, all relevant components can be easily exchanged with other implementations. Besides that, certain services have been already developed and later on been modularized as part of the SpoVNet project, including group communication [2] and a publish/subscribe-service [4]. They are now available as components and will be part of the demonstration.

IV. Our Proposed System

Components in our system are not limited to comply to a fixed set of interfaces, but can rather expose interfaces according to their actually provided functionality. However, there is a standard set of well-defined interfaces of common overlay-related functionality (such as for sending/receiving, multicast, etc.). All components implementing a certain interface are considered to be in one module. As a component can implement more than one interface, it can also be in more than one module. If the functionality of a certain interface is requested, the composition can choose an appropriate component from all the components in that module.

Components are implemented as dynamic libraries (`.so` files in Linux). The composer can calculate a suitable composition at runtime, taking into account the requested functionality and optionally also application requirements. As there is no predefined component interaction, the composition process is not limited to a certain pattern (such as *stacking* components). As components can have arbitrary dependencies, the composition result would be tree-shaped. However, as some components additionally allow to be used simultaneously by multiple parents, a composition is treated and stored as a directed, cycle-free graph (see Figure 2 for two examples). Internally, the boost graph library is used to represent the graph. As a composition graph can be exported/imported as XML, a composition could be also pre-calculate or manually generated, and then simply loaded. The instantiation and linking of the components, however, is always done at runtime, but transparently managed by the framework.

Figure 1 shows an overview over the framework's architecture. The `ServiceComposer` is the central element that can compose and instantiate compositions consisting of any of the available components. Although it is implemented as a static C++ library, it is modularized itself, which means that all core functionality is implemented as exchangeable components as well. The actual setup, i.e., which core components are used,

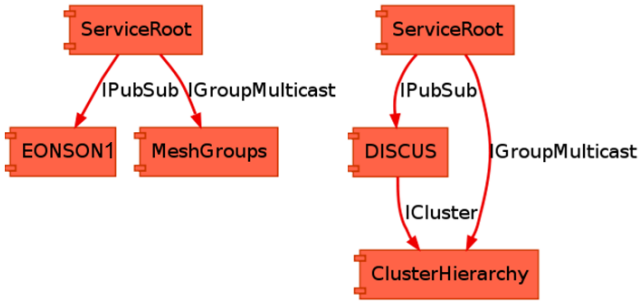

Fig. 2. Two sample composition graphs. In this representation, *ServiceRoot* represents the application itself, which requested the interfaces `IPubSub` and `IGroupMulticast` in this example.

can be configured through an XML file. Currently, three core tasks are modularized into interchangeable components:

A. Component library manager

The task of the component library manager is to tell the composer the set of components that is available for composition. The `LocalXMLFiles` manager, for example, parses a set of local XML files that contain a list of components and their descriptions. A component description consists of the code location (the dynamic `.so`-library), the interfaces a component offers, its dependencies, and other properties. The `ServerRepository` can instead download all required information and code from a central component repository.

B. Composition solver

The solver is the core composition algorithm, which is responsible for resolving all requested interfaces and inter-component dependencies. Depending on the type of algorithm, the solver can also take into account non-functional application requirements. Currently implemented is a `BruteForceSolver`, and a variation that is applying a heuristic to reduce the number of possible compositions. Both also evaluate the possible sharing of a components — this is how a common DHT could be shared among multiple services. To rate different compositions that provide the same service, an experimental evaluation system based on real component benchmarks can be used. A third component that realizes a distributed calculation, utilizing the ariba base communication, is planned.

C. Composition deployment

Usually, the node that is bootstrapping the overlay network is composing the service, which then needs to be deployed to all participating nodes in order for the overlay to function. This is the task of the composition deployment, and currently realized using ariba's base communication.

V. Demonstration Setup & Requirements

The demonstration will show the our framework running dynamic and automatic service compositions. Two applications will use the composer and demonstrate the composition and deployment algorithms. These are a simple chat and a 3D

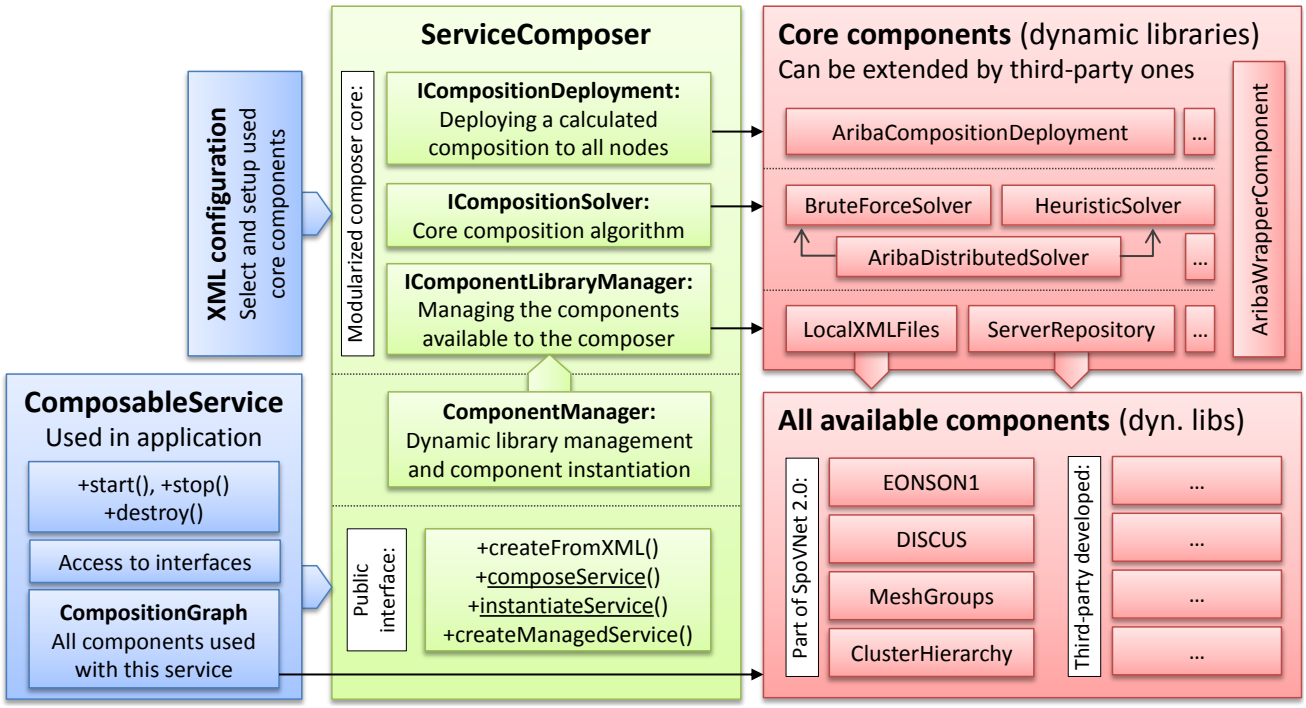

Fig. 1. The composition framework's architecture

multi-player space shooter game called PlanetPI4 (see Figure 3 for a screenshot), which are using the already implemented group communication and publish/subscribe components.

The composition and deployment process will be visualized, which shows how changing optimization parameters does influence the calculated composition, and how this is transferred to all nodes. However, it can also be seen that this is transparent to the application and that all functionality is retained. Further, the automatic deployment of components will demonstrate how missing components can be automatically retrieved.

Fig. 3. The multi-player game PlanetPI4.

VI. Conclusion and Outlook

We propose to demonstrate a component system with automatic composition which is targeted at overlay networks. In contrast to existing overlay composition systems, we provide a highly generic solution, which can be easily extended and adjusted. It comes with an (exchangeable) set of core components, providing algorithms for the composition and component management, as well as deployment techniques. They utilizes the SpoVNet project for overlay-based tools and techniques.

Future work will focus both on meaningful automatic ratings of compositions (with regard to non-functional requirements and component benchmarks) and methods for a P2P distribution of components. For the latter, component integrity and trust are also fundamental issues that require further research.

Acknowledgment

This work was partially funded through the Spontaneous Virtual Networks (SpoVNet) research project by the *Baden-Württemberg Stiftung* within the *BW-FIT* program.

References

[1] R. Bless, C. Hübsch, C. P. Mayer, and O. P. Waldhorst, *Future Internet Services and Service Architectures*, 1st ed. Aalborg, Denmark: River Publishers, Jun 2011, ch. SpoVNet: An Architecture for Easy Creation and Deployment of Service Overlays, pp. 23–47.

[2] C. Hübsch and O. Waldhorst, "Flexible Tree-based Application-Layer Multicast," in *Proceedings of IEEE International Conference on Networks (ICON)*, Singapore, Dec 2011, pp. 159–164.

[3] C. Hübsch, C. P. Mayer, S. Mies, R. Bless, O. P. Waldhorst, and M. Zitterbart, "ariba: Rahmenwerk für Overlay-basierte Dienste," *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, vol. 34, no. 3, pp. 151–155, 2011.

[4] M. A. Tariq, G. G. Koch, B. Koldehofe, I. Khan, and K. Rothermel, "Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints," in *Proceedings of the 16th International Conference on Parallel Computing (Euro-Par 2010)*. Ischia, Italy: Springer, Aug 2010, pp. 458–470.