Design of an Autonomous Software Platform for Future Symbiotic Service Management

Tim De Pauw*†, Nelson Matthys‡, Bruno Volckaert*, Veerle Ongenae†, Sam Michiels‡, and Filip De Turck*
*Department of Information Technology (INTEC), Ghent University – IBBT, 9050 Ghent, Belgium
†Faculty of Applied Engineering Sciences (INWE), University College Ghent, 9000 Ghent, Belgium
‡IBBT – DistriNet, Department of Computer Science, KU Leuven, 3001 Heverlee, Belgium
Email: tim.depauw@intec.ugent.be, nelson.matthys@cs.kuleuven.be

***Abstract*—Nowadays, public as well as private communication infrastructures are all contending for the same limited amount of bandwidth. To optimally share network resources, *symbiotic networks* have been proposed, which cross logical and physical boundaries to improve the reliability, scalability, and energy efficiency of the network as a whole as well as its constituents. This paper focuses on *software services* in such symbiotic networks. We propose a platform for the intelligent composition of services provided by symbiotically connected parties, resulting in novel cooperation opportunities. The platform harvests Semantic Web technology to describe services in a highly expressive manner, and constructs service compositions using SeCoA, our tunable best-first search algorithm. The resulting compositions are then enacted via CaPI, a reconfigurable middleware infrastructure. By means of an illustrative scenario, we provide further insight into the platform's functioning.**

I. Introduction

Over the past few years, home and office environments have seen their numbers of coexistent networks rise dramatically. [1] Many if not all premises nowadays are equipped with several wired or wireless Ethernet networks, ZigBee-based home automation networks, and so forth. Add to that public communication infrastructures based on 3G and 4G, and the result is a veritable overload of networks which are ultimately all contending for the same limited amount of bandwidth. This phenomenon will only be exacerbated, calling for an ingenious approach toward sharing the available resources.

Recently, the research community introduced the concept of *symbiotic networks* [2]. Building upon cognitive networking technology [3], their goal is to allow coexisting networks to communicate transparently across layers and both physical and logical boundaries. This results in cooperation schemes not unlike those of symbiotic organisms. Hence, the envisaged symbiotic networks are highly cooperative and autonomously managed. By handling the available resources more prudently, the aim is to make great strides in terms of scalability, dependability, and energy efficiency, both for the symbiotic network as a whole and for its constituents.

In this paper, we start from the assumption that symbiosis has been established at the infrastructural level, and the networked resources are shared in an efficient manner. In result, the same sort of symbiosis can take place at a higher level. Specifically, *software services* provided by various parties engaging in symbiosis can be shared in a transparent manner as well. By stringing together services which were previously unaware of each other's existence, so-called *service compositions* can expose richer sets of functionality.

However, neither the construction of such symbiotic service compositions, nor their enactment inside the network, is a trivial task. The symbiotic federation of network environments is subject to additional constraints caused by the increased heterogeneity of vocabularies, interaction paradigms, and so forth. Additionally, the dynamicity of symbiotic networks allows for on-the-fly introduction and removal of devices and their services, calling for mechanisms to autonomously relocate and reconfigure software components. In this paper, we propose a software platform which facilitates the construction, enactment, and management of software service compositions in symbiotic network environments.

II. Problem statement

To enable the construction and enactment of service compositions in symbiotically interoperating networks, we envisage research challenges in three main areas.

First, symbiotic environments are subject to vast heterogeneity. Different networks expose different services, with specific APIs, data types, and usage constraints. They may be running on different classes of devices, from resource-rich to resource-constrained, like in the case of *wireless sensor network (WSN)* nodes. To allow for their mixing and matching inside a service composition, there is a clear need for in-network description of services and automated service discovery based upon it.

Second, the software services provided in symbiotic networks each possess their own objectives, be they functional or non-functional in nature. When symbiosis is established, such objectives need to be aligned. Devices and services may however arrive and depart at any time, which may be beneficial or adverse. Autonomous symbiotic service composition mechanisms therefore require periodic feedback from the network.

Third, computed service compositions need to be enacted across all networks involved in the symbiosis. The process of enactment includes a number of activities, such as the mapping of the composition to concrete software artifacts, the installation of these artifacts, and their dynamic reconfiguration. Hence, enactment demands custom middleware solutions which allow the resources involved to be sufficiently dynamic.

III. Solution Architecture

The architecture of the software platform we propose for the construction, enactment, and life cycle management of symbiotic service compositions is visualized in Fig. 1. At the highest level, it consists of two main components.

The first component, *Service Composition*, deals with the descriptions of individual software services provided by symbiotic networks, and uses them to construct compositions. Harvesting *Semantic Web* technology, services and their context are described in a highly expressive fashion. This allows for a versatile *policy* mechanism to govern service interaction.

The produced functional service compositions are passed to the second component, entitled *Composition Enactment*. This component deals with the infrastructural needs of the platform. It translates a composition to a set of interconnected software artifacts, deploys them on the resources at hand, and ensures their proper operation. Vital information pertaining to the infrastructure is passed back to the Service Composition component, giving rise to a feedback loop.

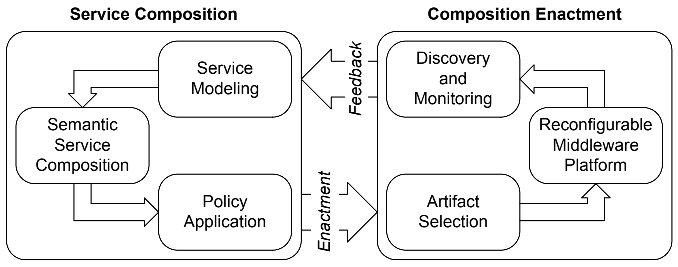

Fig. 1. High-level overview of the platform architecture

IV. Illustrative Scenario

In what follows, we will detail our platform's main components. To further clarify their inner workings, we will apply our approach to an illustrative scenario based on real-life challenges from the field of cargo transport and logistics:

> A warehousing facility stores cargo supplied by several logistics providers, who wish to ensure proper handling of the goods. Specifically, the warehouse offers climate-controlled storage and allows logistics providers to consult the status of their cargo at all times. The cargo itself is fitted with wireless sensor equipment, which exposes the current temperature of the goods along with their unique identification code. Warehouse personnel are informed of the cargo's temperature through a central administration interface, but may also read it from dedicated displays, mounted near the cargo. An external cargo tracking service is periodically updated with additional information provided by the sensor device, allowing the owner of the goods to ensure that they are intact. Communication with this tracking service occurs over the public Internet. Additional measures are therefore taken to maintain the confidentiality of cargo information.

As mentioned, we rely on Semantic Web technology. Specifically, the problem domain is modeled using the *Web Ontology Language (OWL)*, such that service providers can employ the *OWL-S* ontology [4] to describe their offering.

Let us assume that the wireless sensors mounted on the goods expose their temperatures in degrees Fahrenheit. This leads to the introduction of the OWL class FahrenheitTemperature. The warehouse, on the other hand, only understands instances of the CelsiusTemperature class. To describe the information which will need to be provided to the external cargo tracking service, we define the CargoInformation class.

Moving on to a high-level representation of the services involved in the scenario, we now build upon the OWL-S ontology to define the services potentially involved in our compositions, as well as their input and output parameters.

- As is often the case, we assume that the sensor hardware fitted on the cargo can be manufactured by various parties, each with their own software interfaces. Let us define the fictitious sensor device manufacturers *Acme Corp.*, *Sirius Cybernetics*, and *Mom's Friendly Sensor Company*. A sensor device from Acme exposes the AcmeCorpDataProvisioning OWL-S service, with a single output parameter, whose type is the OWL class AcmeCorpSensorMessage. A similar naming scheme is used for the other two manufacturers. One such DataProvisioning service will serve as the *initial service* of the composition.
- Each type of SensorMessage can be analyzed by a specific DataProcessing service. Information from an Acme Corp. sensor, for instance, is fed to the AcmeCorpDataProcessing service, which has two output parameters: a FahrenheitTemperature and a piece of CargoInformation.
- The *goal service* will be CargoQualityAssurance, which takes three confirmation messages as its input. The first, of the type TemperatureVisualizationConfirmation, states that the temperature has been visualized on the dedicated displays. Similarly, a TemperatureVerificationConfirmation confirms that the temperature has been processed by the climate control system, and that appropriate action was taken to ensure the proper temperature for the cargo. Finally, a TrackingConfirmation message states that the tracking service was informed. Respective services entitled TemperatureVisualization, TemperatureVerification, and CargoTracking provide these message types.
- The TemperatureVisualization and TemperatureVerification services only understand the Celsius scale. Thus, their respective input parameters are of the type CelsiusTemperature. To cope with the Fahrenheit temperature readings provided by the sensors on the goods, the warehouse provides a TemperatureConversion service, with transforms its input CelsiusTemperature to a FahrenheitTemperature.
- The CargoTracking service takes CargoInformation as its input. As mentioned, it is external to the warehouse's service infrastructure. To model this, we introduce the OWL class AdministrativeDomain, and the object property belongsTo to link services to domains. This information will be used to construct policies.

V. Service Composition Construction

As outlined in the previous section, our platform's *Service Composition* component manages functional descriptions of services provided in symbiotic networks, and uses them to construct rich compositions. We will clarify our approach by applying it to the illustrative scenario. For a formal description of our composition algorithm *SeCoA*, we refer to [5].

SeCoA constructs service compositions by starting from a *goal service* and satisfying input parameters via backward chaining. The algorithm terminates when the *initial service* has been reached. Let us apply this to our scenario, assuming we are dealing with an Acme Corp. sensor and therefore an AcmeCorpDataProvisioning service.

Our goal service is CargoQualityAssurance, so SeCoA will start by creating a partial composition containing only that service. This composition is placed on SeCoA's priority queue, only to be immediately processed. In each processing step of the algorithm, an attempt is made to satisfy the next remaining input parameter in the composition at the head of the queue.

There are three input parameters in the first partial composition, namely those of CargoQualityAssurance. The first, TemperatureVisualizationConfirmation, is only provided by the TemperatureVisualization service. Therefore, the latter is added to the composition, resolving the input parameter. The resulting partial composition is added to the priority queue and subsequently examined, in the second processing step.

Examining that composition, there are still three unresolved input parameters. One of them is the CelsiusTemperature required by TemperatureVisualization. It can be resolved by prepending TemperatureConversion to that service. Moreover, as TemperatureVerification requires a CelsiusTemperature as well, TemperatureConversion's output can be reused.

Because resolving TemperatureVerification's input did not add any services to the composition, a favorable partial solution was produced. After all, the resulting partial composition has fewer unresolved inputs than any other composition remaining in the queue. This is where SeCoA's *best-first search* comes into play: because of its lower amount of unresolved inputs, the partial composition where TemperatureConversion's output is shared receives a favorable score, and is given priority over other compositions. This behavior can be further influenced by tuning SeCoA's parameters.

Next, the algorithm will search for a service providing either a FahrenheitTemperature or CargoInformation, both of which result in three candidate services: AcmeCorpDataProcessing, SiriusCyberneticsDataProcessing, and MomsFriendlySensorDataProcessing. For all three, a partial composition is placed on the queue. Two of these queue entries will however result in a dead end; for instance, SiriusCyberneticsDataProcessing requires a SiriusCyberneticsSensorMessage input, which is not available. Eventually, the composition containing AcmeCorpDataProcessing will however be encountered and extended with AcmeCorpDataProvisioning. As the latter is the initial service and all input parameters in the composition have been resolved, the algorithm will terminate successfully.

Having constructed a satisfactory service composition, the SeCoA algorithm examines each transfer of a parameter value to see if it matches any of the *policies* defined. This way, non-functional requirements are expressed and satisfied. Policies are written in *SWRL*, the *Semantic Web Rule Language* [6].

In our example scenario, two such policies exist. The first ensures *confidential* communication between the warehouse and the external tracking service. The policy, which relies on our belongsTo object property, looks as follows:

(belongsTo(*Provider*, Warehouse) ∧ belongsTo(*Consumer*, CargoTracking))
∨ (belongsTo(*Provider*, CargoTracking) ∧ belongsTo(*Consumer*, Warehouse))
⇒ apply EncryptionFilter to *ProvidedParameter*
and apply DecryptionFilter to *ConsumedParameter*

Note the use of four SWRL variables, printed in italics, pertaining to the services involved in the parameter exchange.

The second policy states that data processing services must deliver their results with high *accuracy*, by *sampling* measurements over time. Assuming AcmeCorpDataProcessing and its two siblings are subclasses of DataProcessing, the reader familiar with SWRL should be able to devise the policy which applies SamplingFilter to the appropriate input parameters.

Applying both policies results in the filter-augmented graph in Fig. 2, which is also the result of the composition phase.

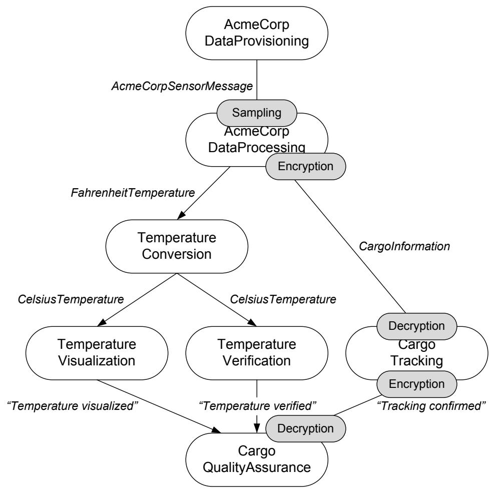

Fig. 2. Service composition produced by applying the SeCoA algorithm to the illustrative scenario described (with policies applied)

VI. Service Composition Enactment

The platform's *Composition Enactment* component supports the composition process by gathering information about the symbiotic network, enacting the compositions constructed by SeCoA by selecting the appropriate software artifacts, and subsequently reconfiguring the underlying infrastructure.

Based on information obtained from the symbiotic infrastructure, compositions produced by the Service Composition component are translated to combinations of software artifacts. A service is not necessarily mapped to a single artifact. For instance, if TemperatureConversion were implemented by a SOAP Web Service, AcmeCorpDataProcessing would need a SOAP client as well as its actual processing logic.

A reconfigurable middleware present on every symbiotic device allows for deployment and run-time reconfiguration of individual software artifacts. We defined *CaPI*, a Component and Policy Infrastructure for networked embedded systems. CaPI is a lightweight, runtime reconfigurable middleware featuring the combination of a *component model*, LooCI [7], and a framework for *policy-based system management*, PMA [8]. In CaPI, symbiotic services are encapsulated through LooCI components, communicating in a loosely coupled fashion with each other via asynchronous events. Components in CaPI can be dynamically deployed, removed, or rewired. Subsequently, PMA policies define an abstraction to separate behavioral concerns of LooCI components from their implementation. Policies are implemented using a declarative policy language featuring an Event-Condition-Action paradigm. At runtime, policies are enforced by intercepting event communication between components, and applying actions to that flow. For example, concerns such as security or reliability can be easily specified using PMA policies and enforced in response. Similar to LooCI components, PMA policies can be dynamically deployed, removed, activated, or deactivated. Finally, CaPI provides mechanisms to discover the set of components and policies present inside every network.

VII. Related Work

Since the emergence of OWL-S, composition of Semantic Web Services has been a popular research topic. Several composition tools have been proposed. [9]–[11] Symbiotic networks, however, introduce an additional set of constraints not foreseen by these utilities, as discussed in Section II. SeCoA attempts to take these constraints into account early, as well as avoid the intermediate model transformations required by some tools. SeCoA policies are currently rather limited; one possible extension would be conflict resolution [12].

Reconfigurable middleware is a critical element for large-scale and long-lived distributed systems. Recently, reconfigurable component models [13], [14] and policy-driven middleware [15] have been introduced to build and manage applications involving resource-constrained devices. However, these approaches typically adopt a tightly coupled style of system composition or lack flexibility in run-time reconfigurability. CaPI approaches the requirements of symbiotic networks by offering more flexibility in run-time reconfiguration via loosely coupled components and dynamically reconfigurable policies.

VIII. Conclusions and Future Work

We introduced a software platform architecture for the autonomous management of service compositions in symbiotic network environments. Relying on Semantic Web technology, highly expressive service profiles are used to construct service compositions, by means of SeCoA, our tunable best-first search algorithm. These compositions are tailored to the infrastructure and subsequently enacted, relying on our reconfigurable middleware platform CaPI. Through a typical scenario, we illustrated key features of our architecture.

In future research, we will be employing the OWL-S ontology's descriptions of service preconditions and effects to create richer compositions. We will also further define mechanisms for mapping high-level compositions to infrastructure-aware software artifact combinations. In particular, we want to clearly quantify the cost in terms of reconfiguration of every combination of artifacts resolved. Hence, we plan a more detailed evaluation of the two types of artifacts CaPI provides.

Acknowledgments

Tim De Pauw would like to thank the University College Ghent Research Fund for financial support through his Ph.D. grant. This research is partially funded by the Interuniversity Attraction Poles Programme of the Belgian State of the Belgian Science Policy, the Flemish Agency for Innovation by Science and Technology (IWT Vlaanderen), and the KU Leuven Research Fund. Part of this work has been funded by the IWT-SBO-SymbioNets project.

References

[1] D. A. Willis, "Hype cycle for networking and communications," Gartner, Inc., Tech. Rep. G00216400, Aug. 2011.

[2] E. De Poorter, B. Latré, I. Moerman, and P. Demeester, "Symbiotic networks: Towards a new level of cooperation between wireless networks," *Wireless Personal Communication*, vol. 45, pp. 479–495, June 2008.

[3] F. H. P. Fitzek and M. D. Katz, *Cognitive Wireless Networks: Concepts, Methodologies and Visions Inspiring the Age of Enlightenment of Wireless Communications*, 1st ed. Springer, 2007.

[4] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith *et al.* (2004) OWL-S: Semantic Markup for Web Services. World Wide Web Consortium. [Online].

[5] T. De Pauw, B. Volckaert, V. Ongenae, and F. De Turck, "SeCoA: Autonomous semantic service composition algorithm in symbiotic networks," in *Proc. IFIP/IEEE Network Operations and Management Symposium (NOMS 2012)*, Maui, HI, USA, 2012, submitted for publication.

[6] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. (2004) SWRL: A Semantic Web rule language combining OWL and RuleML. World Wide Web Consortium.

[7] D. Hughes, K. Thoelen, W. Horré, N. Matthys, P. J. del Cid Garcia, S. Michiels, C. Huygens, W. Joosen, and J. Ueyama, "Building wireless sensor network applications with LooCI," *Journal of Mobile Computing and Multimedia Communications*, vol. 2, no. 4, pp. 38–64, 2010.

[8] N. Matthys, C. Huygens, D. Hughes, J. Ueyama, S. Michiels, and W. Joosen, "Policy-driven tailoring of sensor networks," in *Sensor Systems and Software, Revised Selected Papers. LNICST*, vol. 51. Springer, 2010, pp. 20–35.

[9] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service discovery with OWLS-MX," in *Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems (ACM AAMAS-06)*, Hakodate, Japan, 2006, pp. 915–922.

[10] M. Klusch, A. Gerber, and M. Schmidt, "Semantic Web Service composition planning with OWLS-XPlan," in *Proc. AAAI Fall Symposium on Semantic Web and Agents*, Arlington, VA, USA, 2005.

[11] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau, "HTN planning for Web Service composition using SHOP2," *Web Semantics: Science, Services and Agents on the WWW*, vol. 1, no. 4, pp. 377–396, 2004.

[12] J. Barron, S. Davy, and B. Jennings, "Conflict analysis during authoring of management policies for federations," in *Proc. 1st IFIP/IEEE Workshop on Managing Federations and Cooperative Management (ManFed.CoM 2011)*, Dublin, Ireland, 2011.

[13] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, "A generic component model for building systems software," *ACM Trans. Comput. Syst.*, vol. 26, pp. 1:1–1:42, March 2008.

[14] P. Costa, G. Coulson, C. Mascolo, L. Mottola, G. P. Picco, and S. Zachariadis, "Reconfigurable component-based middleware for networked embedded systems," *International Journal of Wireless Information Networks*, vol. 14, no. 2, pp. 149–162, 2007.

[15] Y. Zhu, S. Keoh, M. Sloman, E. Lupu, N. Dulay, and N. Pryce, "Finger: An efficient policy system for body sensor networks," in *Proc. 5th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2008)*, September 2008.