NeCoMan: Middleware for Safe Distributed Service Deployment in Programmable Networks

Nico Janssens, Lieven Desmet, Sam Michiels, Pierre Verbaeten
DistriNet, Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven, Belgium
{nico.janssens, lieven.desmet, sam.michiels, pierre.verbaeten}@cs.kuleuven.ac.be

ABSTRACT

Recent evolution in computer networks clearly demonstrates a trend towards *complex* and *dynamic* networks. To fully exploit the potential of such heterogeneous and rapidly evolving networks, it is essential for the protocol stacks of the connected devices to adapt themselves at runtime as the environment changes in which they execute. This illustrates the need for employing *programmable* (i.e. adaptable) network nodes. In this paper, we concentrate on deploying *point-to-point based distributed services* in programmable protocol stacks. More in detail, we examine *safe runtime adaptations* of such services so as to preserve service consistency in programmable networks. This has resulted in the development of the NeCoMan (*Ne*twork *Co*nsistency *Man*agement) middleware, a *generic distributed coordination platform* responsible for safe addition, replacement and removal of point-to-point services among programmable nodes. The novelty of this reflective middleware is in its ability to *improve the effectiveness* of the deployment process. This is achieved by customizing the deployment process depending on the properties of both the network service that will be deployed and the underlying execution environment.

Keywords: programmable networks, safe runtime deployment of distributed services, network consistency.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*

1. INTRODUCTION

Research domains such as active networks, ad-hoc networks, ubiquitous computing, pervasive computing and grid computing, clearly demonstrate a trend towards more *complex* and *dynamic* computer networks. This complexity is mainly introduced by (1) unpredictable and varying network link characteristics, caused by unstable wireless communication links, (2) heterogeneous capabilities of connected nodes, and (3) the increasing expectations by users of connected devices towards reliability and quality of service (QoS).

To fully exploit the potential of such complex and dynamic networks, it is essential for the protocol stacks of the connected devices to adapt themselves at runtime as the environment changes in which they execute (e.g. by employing a compression service to boost the quality of a slow wireless connection). To this extent, the underlying protocol stacks should exhibit a similar degree of dynamism, which illustrates the need for employing *programmable* [1] (i.e. adaptable) network nodes. These programmable networks are strongly motivated by their ability to rapidly change the protocol stack of network nodes, without the need for protocol standardization.

This paper focusses on *safe runtime adaptations* of *point-to-point based distributed services* in programmable networks. The point-to-point service model represents a pair of collaborating entities to implement a distributed service. When applied to computer networks, many network services such as compression, fragmentation and encryption conform to this definition.

To prevent runtime adaptation of such services from jeopardizing the correct functioning of computer networks, *safe deployment* is essential. E.g., when a node is extended with a compression component before a decompression module has been installed on its peer, compressed packets cannot be processed correctly. By consequence, the correct functioning of the network is compromised. Performing adaptations of peer nodes at runtime requires a *coordinated deployment protocol* to preserve the integrity of the network while such an adaptation is in progress [10, 16]. Hence, the development of distributed network services for programmable networks is often *complex and error-prone*, especially when additional preconditions (such as QoS, real-time constraints, etc.) are imposed by the execution environment.

To simplify and automate safe deployment of point-to-point based distributed services in programmable networks, this paper presents the NeCoMan (*Ne*twork *Co*nsistency *Man*agement) middleware, which is a *generic distributed coordination platform* responsible for *safe distributed service deployment*[1] at runtime. Stated differently, the NeCo-

[1]The term *safe distributed service deployment* will be used as a collective for safe addition, replacement or removal of distributed services.

3rd Workshop on Adaptive and Reflective Middleware Toronto, Canada

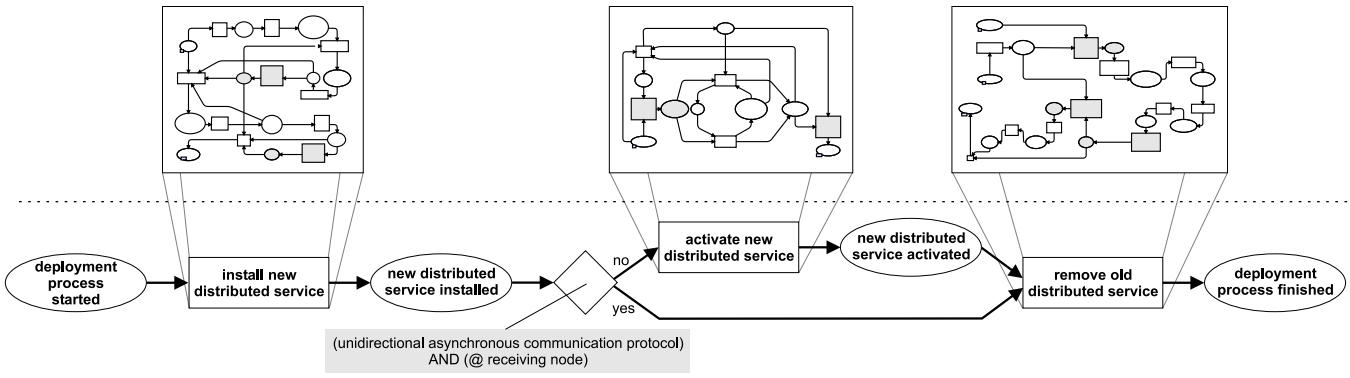


Figure 1: The lower part of this figure illustrates the generic adaptation protocol as employed by the NeCoMan middleware. Depending on the communication model of the point-to-point service that will be deployed, a customized implementation is used at the nodes that are targeted for adaptation. The upper part demonstrates the complexity of the generic adaptation process when taking into account all features that are customizable in the current version of the NeCoMan platform. By packaging this complexity on top of the operating system of programmable nodes, we aim for making the development of network services for programmable networks less complex and error-prone.

Man middleware coordinates transparent and safe addition, replacement and removal of point-to-point services among programmable nodes. By packaging this complexity on top of the operating system of programmable nodes, the developer of a distributed service is spared from implementing a service specific deployment protocol. In this way, we aim to make the development of point-to-point services for programmable networks less error-prone.

Additionally, the NeCoMan middleware has been developed to *improve the effectiveness* of the deployment process, reflecting the properties of (1) the network service that will be deployed and (2) the underlying execution environment. By customizing the deployment process, we aim to minimize the performance penalty of safe distributed service deployment by speeding up the deployment process when possible. Hence, the NeCoMan middleware can automatically select the most effective deployment scenario, depending on inspection of the deployment properties.

The remainder of this paper is structured as follows. Section 2 elaborates on the high-level scenario employed by the NeCoMan middleware to orchestrate safe network service deployment. Next, section 3 illustrates the three most important features that imply customization of the adaptation process so as to improve its effectiveness. Design decisions and related work are discussed in sections 4 and 5, Finally, the paper ends with conclusions and future work in section 6.

2. SAFE DISTRIBUTED SERVICE DEPLOYMENT

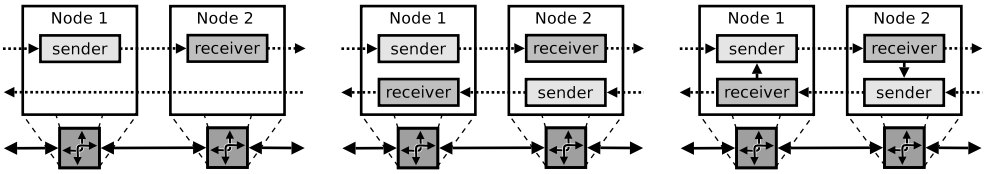
The algorithm to orchestrate safe network service deployment as employed by the NeCoMan middleware comprises three phases: *installation of the new service*, *activation of the new service* and finally *removal of the old service*. This high level scenario is illustrated in the lower part of figure 1.

Installation of the new service. The adaptation process starts with the installation of the new service, resulting in the coexistence of the old service (still in use) and the new version (not yet activated). This is achieved by extending the programmable nodes targeted for service deployment with the new sending and receiving modules implementing the service. In addition, since the old as well as the new service will execute in parallel during the adaptation process, both programmable nodes must be equipped with additional support to distinguish between packets that are handled by both versions. This includes installing support to (1) mark transmitted packets to identify the sending module they were processed by, (2) de-multiplex them when arriving at the receiving node and (3) delegate them to the appropriate receiving module.

Activation of the new service. Next, the newly installed service becomes activated. This only involves the sending node, by *finishing* the old sending module and redirecting new packets to the new version. Since the receiving node is already prepared to handle packets processed by both the old and new sending module, no additional adaptations are required. At this point during the adaptation process, all transmitted packets will be processed by the new service.

Note that safe software reconfiguration requires the software modules that are targeted for adaptation to be both *frozen* and *consistent* [13]. When software modules are consistent, they do not include results of partially completed services. By forcing software modules to be frozen, they (1) are currently not processing any requests and (2) have no pending requests to be accepted and processed. Kramer and Magee define this required consistent and frozen state as the *quiescence* of a component. When *finishing* a software component, it is driven to a state where it complies with this definition [8].

Removal of the old service. Finally, the old distributed service should become removed. Since the old sending module has been finished during the activation phase, it can safely be disconnected. Due to packets processed by the old sending module that might still be in transit across both nodes, the old receiving module should be monitored until it reaches a consistent state [8]. Only when all packets processed by the corresponding sending module are received



(a) Asynchronous unidirectional communication

(b) Asynchronous bidirectional communication

(c) Synchronous communication

Figure 2: Communication models to connect the peer entities of a point-to-point based distributed service.

correctly, the old receiving module can also be removed safely. Finally, since at this point all packets in between both nodes are processed by the new service, the employed support to distinguish between packets that are processed by the old and new service becomes redundant and will be removed.

3. THE NECOMAN PLATFORM

As stated in the introduction, the NeCoMan platform has been developed as reflective middleware. That is, we aim to *improve the effectiveness* of the adaptation process by exploiting the properties of both the network service to be deployed and the underlying execution environment. This section addresses the three most important features that are customizable. Depending on the selected properties, a tailored implementation of the deployment protocol will be used at the nodes that are targeted for adaptation. Due to space restrictions, we can only demonstrate the customizations of the *service installation phase* (see fig. 3(a)). An abstract model of the complete reflective deployment scenario has been illustrated in figure 1.

Service Communication Protocol. The employed communication protocol connecting both collaborating entities that are part of a point-to-point based service, can be either *asynchronous and unidirectional*, *asynchronous and bidirectional* or *synchronous* (as illustrated in figure 2). With respect to the deployment process, this characteristic implies customization of the scenario's responsible for both *installing a new service* (as illustrated in figure 3(a)) and *removing the old service* at the nodes that are targeted for adaptation.

In case of *asynchronous unidirectional communication* (see fig. 2(a)), the point-to-point service that will be deployed only covers the downstream or the upstream path in between two nodes. Such a service is implemented by a sending and a receiving module, each located at different nodes. As an example, we refer to the replacement of a compression service accommodating a slow uplink of a wireless communication channel by a new optimized version.

During the service installation phase, the NeCoMan platform controlling the sending node installs the new compression module as well as support to mark packets that will be compressed by the new module (see fig. 3(b)). Simultaneously, at the receiving node both the new decompression module and support to delegate marked packets to the appropriate decompression module are added to the underlying programmable node (see fig. 3(c)). After this, the receiving node sends a synchronization message to its peer, initiating the activation of the new compression service. Note that only the sending node will execute the activation process (see lower part of fig. 1).

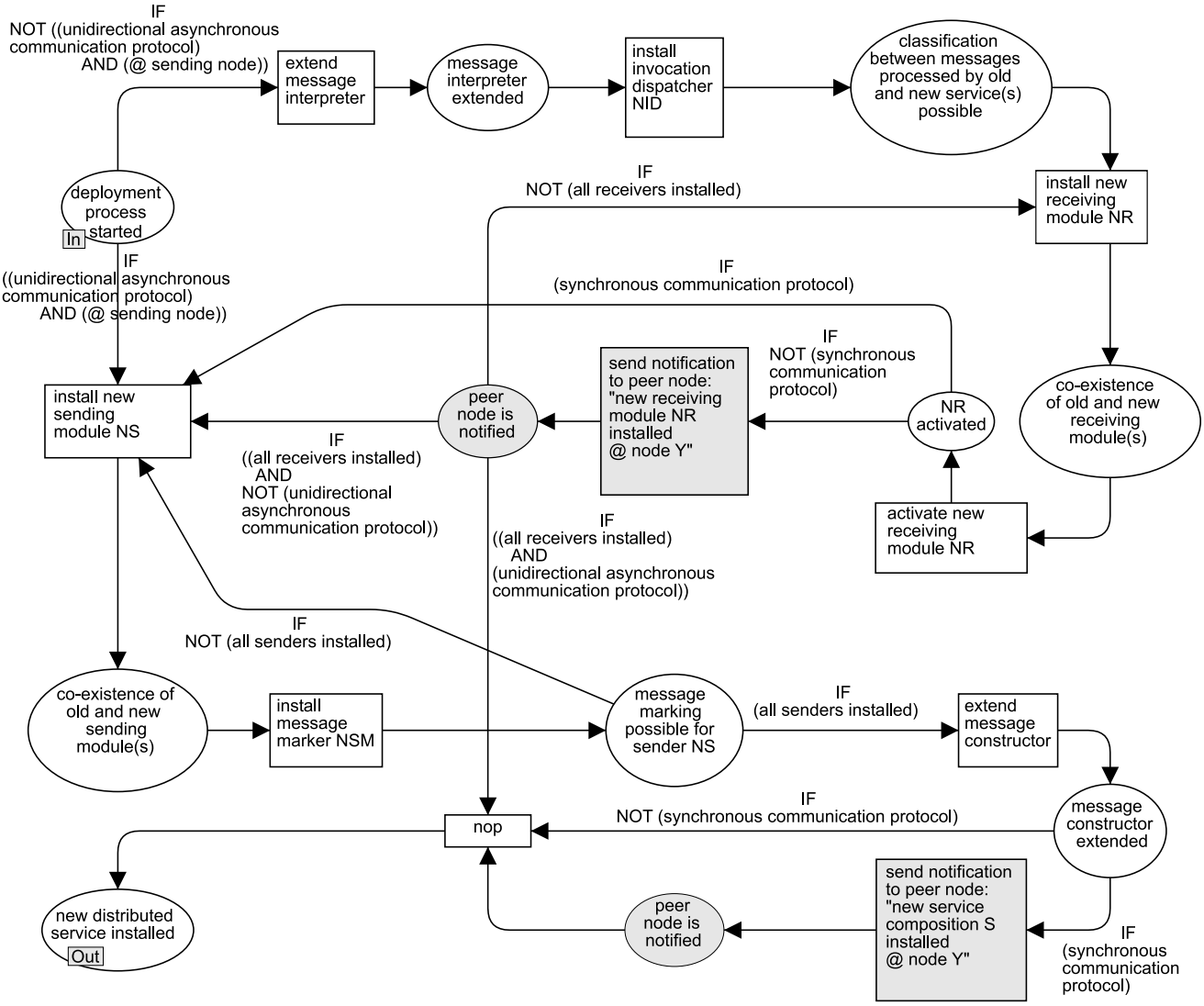
When using an *asynchronous bidirectional communication* protocol (see fig. 2(b)), both the upstream and downstream communication between two routers are *independently* covered by the identical service. In this case, each NeCoMan node controls the deployment of both the *sending part of the downstream protocol* and the *receiving part of the upstream protocol*. As a result, the service installation phase is identical for both nodes (illustrated in fig. 3(d)). In addition, similar to the case of asynchronous unidirectional communication, a synchronization message to activate the new service is sent after installation of the receiving module.

Finally, some network services adopt a *synchronous communication model* (see fig. 2(c)). As an example, we refer to a TCP-like service[2]. In this case, both the downstream (data packets) and the upstream communication (acknowledgements) are part of the same service. This model implies dependencies between the sending and receiving module located at each node. By consequence, (in contradiction to the previous communication models) both the new sender and receiver module will be installed before synchronizing with the peer node to activate the new service (see fig. 3(e)).

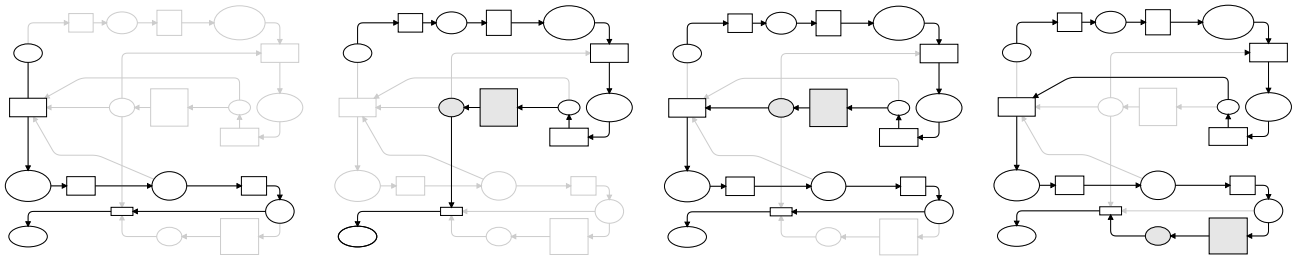
Packet Scrambling. Depending on whether *packet scrambling* could compromise the correct functioning of the network, an adapted implementation of the *service activation phase* will be employed. That is, the order of activating the new sending module and finishing the old version may be altered. On the one hand, if the network does not impose strict packet order guarantees, packets can safely be redirected towards the new sending module while simultaneously finishing the old version. This can speed up the adaptation process, since driving a module to a finished state could come with a performance penalty. On the other hand, when packet scrambling is not tolerated, the old sending module should be finished before the new one is activated.

State Transfer. When no packet scrambling is allowed, *state transfer* between the old (frozen) sending module and the new (not yet activated) version can accelerate the *service activation phase*, depending on the semantics of the network service. For example, consider the replacement of a deployed TCP-like service by a new revised version after the former has successfully completed a 3-way handshake session setup. Rather than waiting for each peer entity to reach the desired

[2]We consider an adapted TCP service that distinguishes between packet losses caused by congestion and transmission errors [5].



(a) The generic model for installing a new service as used by a NeCoMan node. This statechart encapsulates all possible customizations supported by the current version of the NeCoMan platform. The white elements represent the local adaptation activities (i.e. the functional model), while the grey elements reflect the inter-node coordination.



(b) Asynchronous unidirectional service at the sending node

(c) Asynchronous unidirectional service at the receiving node

(d) Asynchronous bidirectional service

(e) Synchronous service

Figure 3: Generic and customized service installation scenarios

CLOSED (finished) state before activating the new service, interrupting (freezing) the running service and conducting the replacement immediately will speed up the adaptation process. However, to preserve a valid service, this implies transferring the internal state of the old TCP service to the new one, before activating the latter. By consequence, the service activation process will be customized depending on (1) the availability of support from the underlying programmable node architecture to capture and reinstate protocol specific state at runtime [6] and (2) the semantics of the network service (stateful vs. stateless).

4. DESIGN

As a proof of concept, the current version of the NeCoMan platform has been implemented in Java on top of DiPS/CuPS [9], a component framework for building runtime adaptable protocol stacks. However, the NeCoMan platform has been developed independently from the underlying node architecture. Adaptations of the underlying protocol stack are expressed in terms of creating, removing, linking and unlinking components from the underlying protocol stack. By consequence, the NeCoMan platform can also be used on top of different adaptable component-based protocol stack architectures, such as Click [12] and NetScript [19].

4.1 Technology background

Since component technology [15] has been illustrated to be a promising means for implementing adaptable middleware, the NeCoMan platform is designed as a *component framework*. Allowing for the customizations as described in the previous section, the NeCoMan design brings together three important research tracks in software engineering research and practice: a *plug-compatible component model*, *pipe-and-filter architectures* and *separation of concerns*:

- As a method for creating maintainable and customizable software, a **pipe-and-filter** architectural style [4] has been proposed. Such an architecture forces a developer to define decoupled components that contain pieces of functionality. These components are plugged one after the other to create a functional system.

- Component frameworks are often referred to as black box frameworks that accept **plug-compatible components** [15]. In order to increase flexibility, NeCoMan components are kept independent from each other by imposing a *plug-compatible component model*. This is achieved by employing a *uniform* pipe-and-filter style, as a result of which all NeCoMan components are equipped with a uniform communication interface. This is a major advantage in terms of flexibility, because it allows individual components to be *reused* in different compositions. The NeCoMan middleware has adopted this architectural style to compose the tailored implementations of the deployment protocol (as illustrated in fig. 3(b)-3(e)), starting from a repository of NeCoMan components.

- **Separation of concerns**. Strict separation of non-functional behavior from the functional code has proven to be an essential feature for building adaptable, maintainable and reusable software [11]. Hence, NeCoMan aims to separate the coordination model from the provided functional model. This allows to customize the employed coordination model of a NeCoMan node without affecting its basic behavior and vice versa [10].

4.2 NeCoMan Component Framework

The research tracks discussed in the previous section have resulted in the design of *NeCoMan components* as first class entities, representing the *transitions* of the deployment scenarios illustrated in fig. 1 and 3(a). The employed plug-compatible component model has resulted in the development of NeCoMan components that are equipped with predefined *incoming* and *outgoing ports*. Communication between two neighboring components is achieved by means of *connectors* (representing the states of the deployment scenario), which are responsible for connecting their incoming and outgoing ports to each other. By consequence, NeCoMan components are kept *unaware* of the counterparts to which they got attached, which improves reuse of NeCoMan components in different compositions implementing customized versions of the NeCoMan middleware.

5. RELATED WORK

In the context of adaptable protocols and distributed services, Chen et al. [2] propose a graceful adaptation protocol that allows adaptations to be made in a coordinated manner across hosts transparently to the application. However, in contrast to the NeCoMan platform, this adaptation protocol is tightly coupled to the underlying node architecture: Cactus [18]. By consequence, we presume it is far from trivial to convert this deployment protocol using different programmable protocol stacks.

Ensemble [17] provides a hierarchical framework for constructing adaptive protocol stacks, where the adaptation is implemented by replacing the entire protocol stack with a new stack. A special Protocol Switch Protocol (PSP) coordinates the adaptation, but since the whole protocol stack is replaced, communication must be stopped until all messages in transit reach their destinations. In contrast, in our approach individual protocol components can be replaced gracefully without interrupting communication.

In order to ensure safety in re-composable software systems, Zhang et al. proposed a method that determines viable sequences of adaptation actions and according safe starting states, based on dependency analysis [7]. Since all determined adaptation actions of their method can be considered as small service deployments in a distributed environment, the approach presented in this paper is complementary to their method and can be combined in a more general safe adaptation technique for larger systems.

Finally, [14] and [3] present research in the middleware context, tackling the problem of adapting distributed applications by means of reflective and adaptive middleware. However, in contrast to our work, these systems do not provide in tailored deployment scenarios, taking into account non-functional preconditions (such as QoS, real-time constraints, etc.) that are essential in the context of programmable networks.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the NeCoMan middleware, a *generic and reflective distributed coordination platform* responsible for transparent and safe addition, replacement and removal of point-to-point services among programmable ad hoc nodes. In addition, we have illustrated how to

improve the effectiveness of the deployment process taking into account (1) the employed service communication protocol connecting both collaborating entities that are part of the point-to-point based service, (2) whether packet scrambling could compromise the correct functioning of the network and (3) whether support for state transfer is provided. Due to space restrictions, we have only presented the customizations of the *service installation phase* (see fig. 3(a)). An abstract model of the complete reflective deployment scenario has been illustrated in figure 1.

Unresolved problems. Future research involves the investigation of three important sub-tracks:

First of all, tool support for automatically composing the appropriate NeCoMan composition given the properties of both the network service to be deployed and the underlying execution environment is under development.

Secondly, we intend to formally validate the correctness of the deployment process as used by the NeCoMan platform.

Next, we will extend the NeCoMan middleware such that distributed services consisting of *several collaborating entities* (rather than only point-to-point based services) can safely be deployed in programmable networks. We believe this boils down to employing a different coordination model, taking into account all participating nodes. Since the coordination model of the NeCoMan middleware has been separated from the functional model, employing a different coordination model will not affect the local adaptation activities offered by the current version of the NeCoMan middleware [10].

Finally, we will extend the presented deployment scenario with roll-back support, to be applicable in ad-hoc networks. These networks are characterized by frequent disconnections as well as autonomous mobility of mobile devices, which results in a highly dynamic network topology.

7. ACKNOWLEDGMENTS

Research for this paper has been carried out with financial support of the Fund for Scientific Research Flanders, Belgium – F.W.O. RACING #G.0323.01

8. REFERENCES

[1] A. T. Campbell, H. G. De Meer, M. E. Kounavis, K. Miki, J. B. Vicente, and D. Villela. A survey of programmable networks. *SIGCOMM Comput. Commun. Rev.*, 29(2):7–23, 1999.

[2] W.-K. Chen, M. Hiltunen, and R. Schlichting. Constructing Adaptive Software in Distributed Systems. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS'02)*, pages 635–643. IEEE Computer Society.

[3] G. Coulson, G. Blair, A. T. Gomes, A. Joolia, K. Lee, J. Ueyama, and Y. Ye. A Reflective Middleware-based Approach to Programmable Networking. In *The 2nd Workshop on Reflective and Adaptive Middleware*, Rio de Janeiro, Brazil, 2003.

[4] D. Garlan and M. Shaw. An introduction to software architecture. In *Advances in Software Engineering and Knowledge Engineering*, volume 1. World Scientific Publishing Co., 1993.

[5] J. Hoebeke, T. V. Leeuwen, L. Peters, K. Cooreman, I. Moerman, B. Dhoedt, and P. Demeester. Development of a TCP protocol booster over a wireless link. In *Proceedings of the 9th Symposium on Communications and Vehicular Technology in the Benelux (SCVT 2002)*, Louvain la Neuve, oct.

[6] C. Hofmeister. *Dynamic Reconfiguration of Distributed Applications*. PhD thesis, Department of Computer Science, University of Maryland, Jan. 1994.

[7] B. H. C. J. Zhang, Z. Yang and P. K. McKinley. Adding safeness to dynamic adaptation techniques. In *Proceedings of Third Workshop on Architecting Dependable Systems (WADS 2004)*, pages 17 – 21, Edingburgh, Scotland, May 2004.

[8] N. Janssens, S. Michiels, T. Holvoet, and P.Verbaeten. A Modular Approach Enforcing Safe Reconfiguration of Producer-Consumer Applications. In *Proceedings of The 20th IEEE International Conference on Software Maintenance*, Chicago Illinois, USA, September 2004.

[9] N. Janssens, S., Michiels, T. Mahieu, and P. Verbaeten. Towards Hot-Swappable System Software: The DiPS/CuPS Component Framework. In *Proceedings of The Seventh International Workshop on Component Oriented Programming*, 2002.

[10] N. Janssens, E. Steegmans, T. Holvoet, and P. Verbaeten. An Agent Design Method Promoting Separation Between Computation and Coordination. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 456–461. ACM Press, 2004.

[11] G. Kiczales. Towards a New Model of Abstraction in the Engineering of Software. In *Proceedings International Workshop on New Models for Software Architecture (IMSA): Reflection and Meta-Level Architecture*, Tokyo, Nov. 1992.

[12] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.

[13] J. Kramer and J. Magee. The evolving philosophers problem: Dynamic change management. *IEEE Transactions on Software Engineering*, 16(11):1293–1306, 1990.

[14] F. J. S. Silva, M. Endler, and F. Kon. A Framework for Building Adaptive Distributed Applications. In *The 2nd Workshop on Reflective and Adaptive Middleware*, Rio de Janeiro, Brazil, 2003.

[15] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Publishing Co., 1998. ISBN 0-201-17888-5.

[16] E. Truyen, B. Vanhaute, W. Joosen, and P. Verbaeten. Consistency Management in the Presence of Simultaneous Client-Specific Views. In *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, pages 501–510, Oct. 2002.

[17] R. van Renesse, K. Birman, M. Hayden, A. Vaysburd, and D. Karr. Building adaptive systems using ensemble. *Software – Practice & Experience*, 28(9):963–979, 1998.

[18] G. T. Wong, M. A. Hiltunen, and R. D. Schlichting. A configurable and extensible transport protocol. In *Proceedings of INFOCOM 2001*, pages 319–328, 2001.

[19] Y. Yemini and S. daSilva. Towards programmable networks. In *IFIP/IEEE International Workshop on Distributed Systems:Operations and Management*, Oct. 1996.