Towards composition of distributed evolving services: the Credo approach (Invited Paper)

Andries Stam
Almende BV
Westerstraat 50
Rotterdam, The Netherlands
andries@almende.com

Alfons Salden
Almende BV
Westerstraat 50
Rotterdam, The Netherlands
alfons@almende.com

ABSTRACT

ICT service providers face increasing demands on dynamic, flexible and scalable composition of their evolving software services. These demands complicate the validation and verification of such compositions as–a–whole. Within the European *Credo* research project, we develop techniques for the modeling, validation and verification of compositional distributed services. Our approach is based on two principles: a clear formal separation between the service components and the logical network that binds them together, and support for light–weight, preferably automated verification and model checking for all modeling techniques. In this paper, we apply the *Credo* techniques to ASK, a context–aware response system with intelligent matching functionality for connecting people to other people via existing communication technologies.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Formal Methods; D.2.7 [**Distribution, Maintenance, and Enhancement**]: Restructuring, reverse engineering, and reengineering

Keywords

services, composition, evolution, exogenous coordination, *Credo*, Creol, Reo, Automata, C

1. INTRODUCTION

ICT Service providers face increasing demands on dynamic, flexible and scalable composition of their software services. Customers more and more require that ICT services of various suppliers can be composed and dynamically adapted on demand, while the resulting compositions should continuously ensure certain levels of availability and robustness. If the service components themselves *evolve* over time, this introduces additional complexity to the verification of functional and non–functional properties of their compositions. In the European FP6 project *Credo* (IST–33826), we develop techniques and tools for the modeling, validation and verification of compositional distributed services. The techniques and tools are based on two principles: a clear formal separation between the service components and the logical network that binds them together, and support for light–weight, preferably automated verification and model checking for all modeling techniques.

In this paper, we provide a survey of the work that is being carried out in the context of the *Credo* project, focusing on the composition of distributed evolving service components. We apply the *Credo* techniques to ASK, a context–aware response system with intelligent matching functionality for connecting people to other people via existing communication technologies. We show that the principles of *Credo* enable the modeling and verification of the ASK system during evolution of its distributed service components.

The paper is organized as follows. In Section 2, we introduce the *Credo* approach together with its modeling and verification techniques. In Section 3, we give a short overview of the ASK System as it exists currently, and indicate in which directions we would like to evolve it. In Section 4, we show how the *Credo* tools are currently being applied to ASK, in order to transform this system into a set of small distributed service components which can be dynamically composed to cope with changing functional and non–functional requirements. Finally, in Section 5, we conclude and present future work.

2. THE CREDO APPROACH

The *Credo* Project aims at the development of formal methods and techniques for the modeling and analysis of compositions of distributed evolving services. A basic principle in *Credo* is the adoption of a clear separation of concerns between the components providing the services and the logical network via which these components are connected. This separation of concerns is at least adopted at the level of modeling and analysis, but it can also be realized at the level of implementation, if desired. Separate modeling languages and light–weight or automated verification techniques and tools are developed for the components, the network, and their combination. Thereby, *Credo* focuses on the analysis of the effect of component changes and reconfiguration of the network.

An overview of the primary entities recognized in service component models is shown in Figure 1. *Service components* are used as the container entities for services. Their functionality is modeled in terms of *classes*, instantiated into *objects*. A composition of service components is connected via a logical *network*, which determines how communication between the components takes place in order to let them work together as a system. Both components and their inner classes have so–called *behavioral interfaces*, which specify their behavior and non–functional behavioral properties in an abstract way. They can be used for the verification of compositions of either objects or components. In the next subsections, we introduce the *Credo* modeling languages used for components, objects and the network and briefly mention the supported verification techniques.

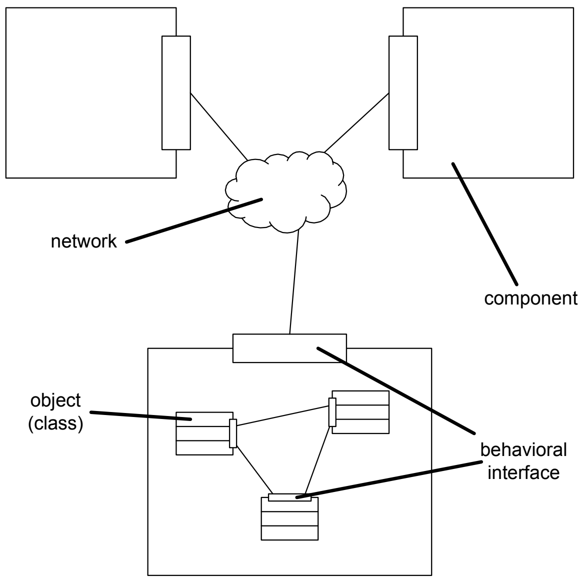

Figure 1: End user perspective overview

2.1 Components and Classes: Creol

Service components and classes are modeled using the statically typed high–level concurrent object–oriented modeling language Creol [12, 13]. The expressiveness of the Creol language makes it very suitable for modeling, since in many cases models of a system can be structured in a way very similar to the implementation of that system (e.g. with respect to method names and flow of control). Creol has a formal syntax and concise operational semantics. To give the reader an idea of how models in Creol are specified, Figure 2 provides a simple example of a Creol *interface* and *class*, the latter one implementing a single method outputting the sum c of two integers a and b.

Tools are available to edit, type check, compile and run Creol models from Eclipse. Run–time checking of assertions and model checking can be done by running Creol models in the rewrite engine Maude [9]. An important property of Creol is that type checking can be performed in the context of *dynamic class updates* [19]. Currently, we investigate ways to test and verify system implementations written in the programming language C against Creol models.

```
interface Adder
begin
  with Any op add(in a: Int, b: Int; out c: Int)
end

class Adder implements Adder
begin
  with Any op add(in a: Int, b: Int; out c: Int) ==
    c := a + b
end
```

Figure 2: Example of a Creol Model

2.2 The Network: Reo

Component compositions are realized by connecting sets of components via logical networks. For the modeling and realization of such networks, *Credo* provides several techniques and languages, of which the most important one is the channel–based exogenous coordination language Reo [4, 3]. In Reo, communication between components is modeled in terms of *channels* connected to each other via *nodes*. Components exchange values via *ports*, which are connected to certain Reo nodes. An important enabler for the dynamic composition of distributed evolving service components is the fact that Reo is an *exogenous coordination language*: components communicate solely via ports, remaining completely unaware of the identity of other components in their environment. The values they send or receive have no target or source address. Instead, it is the responsibility of the logical network, the *Reo circuit*, to route messages between service components.

The example of Figure 3 shows a Reo circuit connecting three service components. This circuit ensures that values coming from the left port of component A are received by either component B or component C, depending on the availability of a value on one of the ports at the right side of A. This simple circuit already illustrates how dynamic reconfiguration for purposes of e.g. load balancing can be realized without changing the components used in the composition.

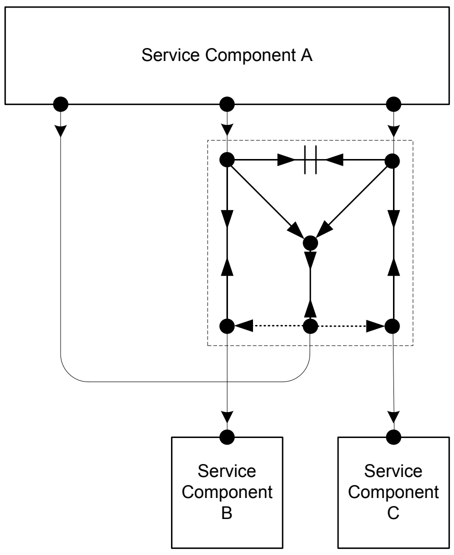

Figure 3: Example of a Reo Network

Reo editors, animators and model checking tools are available as plug–ins for Eclipse. Various extensions for Reo are available as well, like tools to check the equivalence of Reo networks [8] and the integration with well–known business process languages like BPEL. Currently, progress is made regarding dynamic reconfiguration of Reo circuits and automated code generation for the programming language C.

2.3 The Integration: Automata

An important issue in the composition of distributed evolving service components is the validation and verification of the behavior of a composition *as–a–whole*. In *Credo*, light–weight verification is possible through the usage of *Behavioral Interfaces*. Such interfaces describe the abstract behavior of a class or a component in terms of the possible *sequences of communication* with them, for example in terms of inputs (dependencies) and outputs (results). For their specification, we use several types of *automata*, largely inspired by the idea of *interface automata* [10].

Automata provide a concrete and intuitively clear model of computation, and a structural approach to the analysis of the behavior of components and their composition. Multiple automata can be combined via product functions into larger automata. Checks can be performed whether, given a certain abstraction function, the behavioral interfaces for a set of objects in combination realize the behavior specified in the behavioral interface for a component. More importantly, the compatibility of behavioral component interfaces and Reo networks can be checked, via the generation of so-called *constraint automata* for Reo circuits [5]. Progress is made within *Credo* regarding tools for the model checking of *timing constraints* and *scheduling policies* for components and objects.

Two simple examples of behavioral interfaces in terms of automata are given in Figure 4. The *transitions* between the *states* in the automata specify value inputs (?) or outputs (!) on certain *ports* (x, y, z, z'). Transitions from a black dot indicate initial states. The automaton at the left of the figure specifies that a component or object implementing this automaton repeatedly behaves as follows: Firstly, receive a value via port x. After that, send a value via either port y or port z. The syntax of automata can be easily extended to allow for the specification of various non–functional properties on the transitions, like timings or memory consumption.

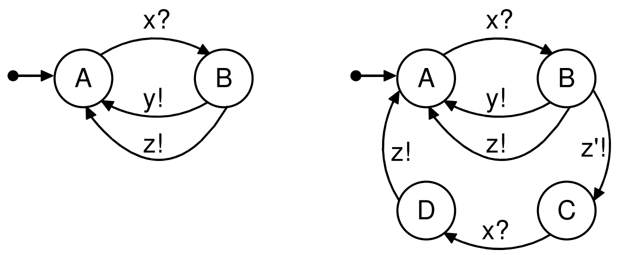

Figure 4: Behavioral Interface Automata

The concept of behavioral interface is one of the keys to the verification of compositions of distributed evolving services. Once a service component evolves, this likely causes changes to its behavior at the interface level, or to the value types exchanged through its ports, or to its non–functional properties. Hence, newer versions of its behavioral interfaces can be used to incorporate this evolution in an abstract way. These newer versions, in turn, can then be used *a priori* or *a posteriori* to verify whether the composition as–a–whole, including its Reo circuit, satisfies composition–level requirements.

In the next section, we introduce the ASK system, which is used as a case study for the *Credo* “approach” presented above. we show some initial results in the application of this approach to ASK in Section 4.

3. THE ASK SYSTEM

ASK has been developed by Almende [1], a Dutch research company focusing on the application of self–organisation in human organisations and agent–oriented software systems. The system is marketed by ASK Community Systems [2]. ASK provides mechanisms for matching users requiring information or services with potential suppliers. Based on information about earlier established contacts and feedback of users, the system learns to bring people into contact with each other in the most effective way. Typical applications for ASK are workforce planning, customer service, knowledge sharing, social care and emergency response. Customers of ASK include the European mail distribution company TNT Post, the cooperative financial services provider Rabobank and the world’s largest pharmaceutical company Pfizer. The amount of people using a single ASK configuration varies from several hundreds to several thousands.

3.1 Purpose and Functionality

The primary goal of the ASK system is to connect people to other people in the most effective way. The system acts as a *mediator* in establishing the contacts: people can contact the system via various media like telephone or email, and the system itself is also able to contact people via those media. In determining the *effectiveness* of contact establishment, multiple aspects play a role. For example, the rating of *human knowledge and skills* is important in cases where people request contact with specialists or service providers. In these cases, the ASK system is able to ask participants for feedback on the quality of service after the contact. This feedback can be used for optimization of subsequent requests of the same kind. A different role is played by *time schedules*, which indicate when certain people can be reached for certain purposes. The ASK system differentiates between regular plannings and ad–hoc schedules caused by sudden events or delays. Different *communication media* play another role. In most ASK configurations, voice communication (phone, VoIP) is the primary communication medium used, but different media like email and SMS are supported by ASK as well. Moreover, people can own various phone numbers and email addresses, for which they can indicate preferences and time or service dependent usage constraints. The ASK system is able to exploit knowledge about the reacheability of people via specific media, for example in the context of emergency response systems, where people must be contacted within a certain time window. In general, learning from past experiences of all kinds and forecasting based on these experiences plays a crucial role in ASK.

3.2 Technical Architecture

The software of ASK can be technically divided into three parts: the *web front-end*, the *database* and the *ASK engine* (see Figure 5). The *web front-end* acts as a configuration dashboard, via which typical domain data like users, groups, phone numbers, mail addresses, interactive voice response menus, services and scheduled jobs can be created, edited and deleted. This data is stored in a *database*, one for each configuration of ASK. The feedback of users and the knowledge derived from earlier established contacts are also stored in this database. Finally, the *ASK engine* consists of a quintuple of components *Reception*, *Matcher*, *Executer*, *Resource Manager* and *Scheduler*, all written in the C programming language, which handle inbound and outbound communication with the system and provide the intelligent matching and scheduling functionality.

The mechanism of dynamic reconfiguration has been promoted by Almende in earlier research on the Common Hybrid Agent Platform (CHAP) [18], of which one part, the *Abbey*, is the foundation for each of the components in ASK. An abbey is an *enhanced thread pool*, in which separate threads, called *monks*, are capable of performing arbitrary *tasks*. Components are able to communicate with each other by sending and/or receiving *requests* as part of a task. Hence, the structural facilities for reconfiguration, in terms of independently executable tasks, are already present in the architecture.

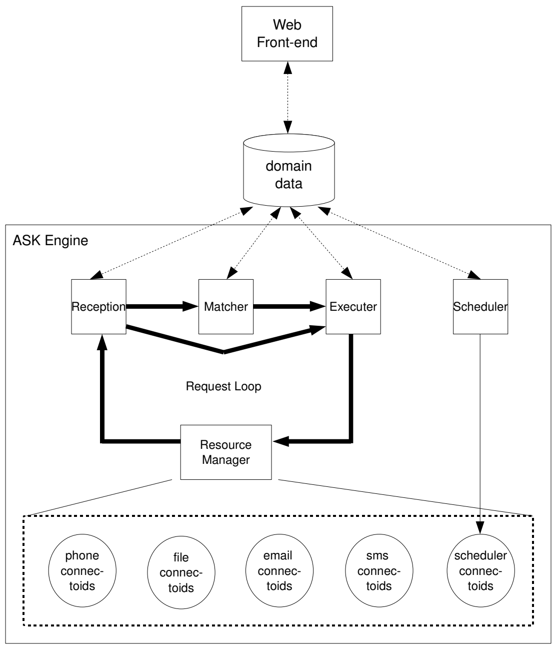

Figure 5: ASK System Overview

The "heartbeat" of the ASK engine is the *Request loop*, indicated with thick arrows. Requests loop through the system until they are fully completed. The *Reception* component determines which steps must be taken by ASK in order to fulfil (part of) a request. The *Matcher* component searches for appropriate participants for a request. The *Executer* component determines the best way in which the participants can be connected. ASK clearly separates the medium and resource independent request loop from the level of media–specific resources needed for fulfilling the request, called *connectoids* (e.g., a connected phone line, a sound file being played, an email being written, an SMS message to be sent). The *Resource Manager* component acts as a bridge between these two levels. Finally, a separate *Scheduler* component schedules requests based on job descriptions in the database. In the next paragraphs, we discuss in more detail those components which create and exploit knowledge in ASK: the *Reception*, the *Matcher* and the *Scheduler* component.

Reception.

The major role of the Reception component is to determine which action should be taken by the ASK system based on a request. To give an example, if a request is received containing an incoming call event from a certain telephone number, the Reception component can decide to present a specific interactive voice response (IVR) menu to the caller, depending on the current date and time, number of the caller and the number being called. The caller is then able to provide information about the request, by selecting submenus or actions via dual–tone multi–frequency (DTMF) dial tones. A request could also originate from the scheduler, for example if the ASK system calls a user in order to ask for feedback or for availability as an ASK responder for a certain time period. The reception component is responsible for performing updates to the contents of the database in terms of adding previously unknown telephone numbers, adding feedback from users or changing schedules of responders.

Matcher.

The Matcher component tries to find users satisfying a request. For example, a person calling the ASK system could ask for a connection with a specialist on a certain topic. Matching can be complicated, since the preferences and time schedules of the requester and candidate responders must be taken into account, as well as feedback about earlier contacts. The Matcher tries to find several candidate responders and selects between them using one of four possible methods:

1. Round Robin: the Matcher randomly selects a responder from the set of candidates available.

2. Last Spoken: the Matcher selects the responder that was selected previously.

3. Rating: the Matcher uses feedback provided by the requester about potential responders and selects the one with the highest rating.

4. Friendly Rating: the Matcher again selects based on the received ratings, but occasionally randomly selects a different responder in order to provide them with the opportunity to improve their rating.

Scheduler.

The Scheduler component realizes the execution of various types of scheduled jobs. Typical jobs are: contacting requesters and responders to obtain feedback about earlier connections, or contacting potential responders for availability. In executing these jobs, the Scheduler component keeps track of the time schedules and preferences of users. The Scheduler itself does not take part in the request loop: its messages enter the request loop as if they come from outside the system. Jobs for the Scheduler can be put into the database manually via the web front–end, or automatically, as the result of the execution of requests in the ASK engine.

3.3 Future Developments of ASK

Currently, ASK configurations are deployed on a per–customer basis in a centralized manner. Developments in information and communication services, however, call for more openness and distribution of the services of ASK. Future developments of ASK likely include the following:

- improving *context–awareness and sustainability* of the ASK system, with a focus on distribution, replication and adaptation of its components and component interactions.
- improving *synthesis* of ASK and other systems, with a focus on enabling and coordinating the exchange of data from different domains. Currently, we investigate synthesis with logistic systems.
- improving *organization–specific customization* and *personalization* of the ASK system, with a focus on data distribution, data encapsulation and agentification.

Especially the first topic is relevant in the context of this paper. In the next section, we will show how we apply the *Credo* modeling and verification techniques for improving distribution, replication and adaptation of the ASK system.

4. APPLYING CREDO TO ASK

As we explained, the basic modeling entities in the *Credo* approach are Creol models for service components, Reo circuits for their exogenous coordination networks, and automata for behavioral interfaces.

4.1 Creol and Automata

We use Creol to model the functionality of the ASK system, at a high level of abstraction, primarily for purposes of *analysis and verification.* Automata are used to capture the behavioral interfaces of components and of certain important separable inner functions of these components. The Creol models can be verified against these automata, which provides us two different levels of abstraction on top of the actual code. Methods and techniques are currently developed to verify Creol specifications against the C codebase of ASK.

As we explained in Section 2, automata specifications of behavioral interfaces are especially useful for purposes of evolution, for three reasons. Firstly, they capture the *dynamics* of service components, whichs allows for *compatibility checking* of evolving service components against a network, and *model checking* of a set of automata in combination. Secondly, the types of exchanged values can be specified, which allows for static *type checking.* In the context of evolution, we envision extensions to the type checking approach, in terms of meta-information about complex types, or simple filtering and transformation techniques for purposes of type conversion. The approach of Reo, with its extensible set of channels, is particularly useful for such a latter extension. Thirdly, non–functional properties relevant in the context of application can be modeled with automata as well. To give an example: because time and scheduling plays a considerable role in the ASK system, we use *timed automata* [6] and their derivative *task automata* [11] to model, simulate and model check timing issues in UPPAAL, an integrated tool environment for the modeling and simulation of real–time systems.

For the near future, we have planned to experiment with the integration of automata and Creol models *within* service components, as a means to provide them an abstract *self–model.* These models can be used within a composition of evolving service components for a priori compatibility, model or type checking. Although the checks themselves are currently still carried out by hand, we expect that at least parts of these checks can be completely automated.

As an example, Figure 6 shows a service component in three stages of evolution. In its first stage, it provides interface *old interface* and *meta–interface.* The meta–interface provides ways to access the automata and Creol models of the service component. In an intermediary stage, a *new interface* is added. As a consequence, the meta–interface now provides new or updated models, which can be used to check how the new or updated interfaces can be exploited and what kind of dependencies they introduce in the composition. If exploitation is possible and the dependencies can be fulfilled, the Reo circuit can be adapted to incorporate the new interface in the composition, for example through a reconfiguration mechanism like shown in Figure 3. Eventually, if no use is made of the old interface anymore, this interface could be removed and we enter the evolved third stage.

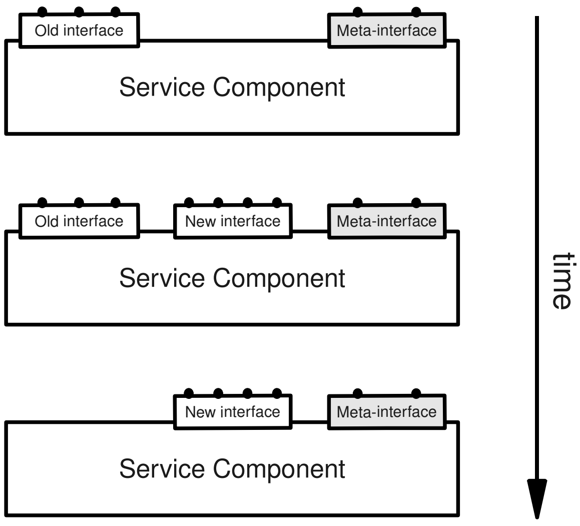

Figure 6: Evolution of Component Interfaces

4.2 Reo

Communication within and between ASK components is reorganized into communication via new C code generated from Reo circuit specifications, as a means to achieve better compositionality through exogenous coordination and to enable dynamic composition and reconfiguration. Exogenous coordination is an important enabler of dynamic reconfiguration. Take for example Figure 7, which represents the core ASK components connected to each other via Reo channels. Suppose, for example, that we would like to add a new component with new matching functionality, say *Matcher'.* Exogenous coordination makes it possible to easily reroute requests from the Reception component to either the Matcher component or the Matcher' component, without any changes to the Reception component *itself.*

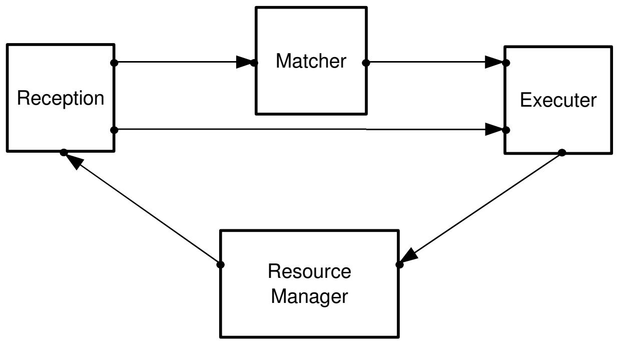

Figure 7: Exogenous Communication in ASK

Furthermore, we plan to reorganize the inner structure of the ASK components as well. In fact, we regard the components as *compositions* of even smaller components. An example is shown in Figure 8, which illustrates how the Reception component can be internally structured as a composition of a *Case Selector*, which determines the type of an incoming request, and a set of *Request Handlers* with knowledge of how a particular request can be fulfilled. Again, Reo circuits are used to connect the smaller components to each other. This shows that the same composition mechanism can be used at different levels of service granularity, allowing for a hierarchy of compositions.

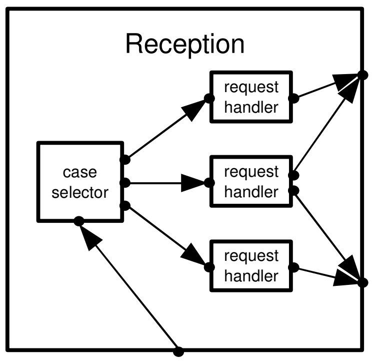

Figure 8: Exogenous Comm. inside Components

Clearly, Reo is an important enabler for all kinds of dynamic reconfiguration, while Creol and automata enable us to reason about the properties of specific configurations. Thereby, the *Credo* approach provides the necessary ingredients for reasoning about compositions of evolving distributed service components.

5. CONCLUSIONS AND FUTURE WORK

We have pointed out the added value of the *Credo* approach for robust and flexible adaptation of compositions of distributed evolving services. We showed how we currently apply the modeling techniques of *Credo* to the ASK system: we create Creol and automata models for the ASK service components in order to reason about their individual behavior and non–functional properties, and we use Reo circuits to connect the components together and to coordinate them in an exogenous manner. Thereby, we are able to implement all kinds of reconfigurations, which can be verified via Creol and automata models, and actuated via Reo circuits.

In future work we will focus on the evaluation of the *Credo* approach in enabling the synthesis of ASK with other systems and the personalization and customization of ASK. In this respect, we would like to further develop the idea of using *Credo* models as *self-models* for service components. Another challenge is to enhance the *Credo* modeling languages with constructs for semantic modeling.

As an engineering challenge, we foresee finding the right tools for the identification of categories of models of evolvable service components. Several models and infrastructures were proposed in [17, 15, 7, 14] for grounding formal modeling languages, like those of *Credo*, and semantic web tools, on the basis of evolving business networks. Such models and infrastructures leverage dynamic service management and adaptation in a robust, distributed and sustainable way. They help in identifying the appropriate boundaries of service components. Furthermore, they also capitalize on these identified boundaries by inducing potential semantics–oriented and cognitive network categories. Such network categories are assumed to emerge upon self–organization and form the basic reusable modular structures and functional organization schemes for self–management and self–adaptation [16]. In the context of this paper it may also be interesting whether the emerging self–management and self–adaptation modules can be represented in terms of the *Credo* system, i.e. as living instances of Creol models, automata and Reo circuits. In addition the question arises whether the theories of self–organization can shed a light on what is missing in the *Credo* modeling languages and verification techniques for the representation and checking of functional and non–functional service requirements at run–time.

6. REFERENCES

[1] Almende website. `http://www.almende.com`.
[2] ASK community systems website. `http://www.ask-cs.com`.
[3] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical. Structures in Comp. Sci.*, 14(3):329–366, 2004.
[4] F. Arbab and F. Mavaddat. Coordination through channel composition. In *COORDINATION '02: Proceedings of the 5th International Conference on Coordination Models and Languages*, pages 22–39, London, UK, 2002. Springer-Verlag.
[5] C. Baier, M. Sirjani, F. Arbab, and J. Rutten. Modeling component connectors in reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006.
[6] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.
[7] G. Binnig, M. Baatz, J. Klenk, and G. Schmidt. Will machines start to think like humans – artificial versus natural intelligence. *Europhysics news*, pages 44–47, 2002.
[8] T. Blechmann and C. Baier. Checking equivalence for reo networks. *Electron. Notes Theor. Comput. Sci.*, 215:209–226, 2008.
[9] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada.

Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001. This volume.

[10] L. de Alfaro and T. Henzinger. Interface automata. In *Proceedings of the.* ACM Press, January 2001.

[11] E. Fersman, P. Krcal, P. Pettersson, and W. Yi. Task automata: Schedulability, decidability and undecidability. *Inf. Comput.*, 205(8):1149–1172, 2007.

[12] E. B. Johnsen and O. Owe. An asynchronous communication model for distributed concurrent objects. *Software and Systems Modeling*, 6(1):35–58, Mar. 2007.

[13] E. B. Johnsen, O. Owe, and I. C. Yu. Creol: A type-safe object-oriented model for distributed concurrent systems. *Theoretical Computer Science*, 365(1–2):23–66, Nov. 2006.

[14] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer Society*, pages 41–50, 2003.

[15] P. Oreizy, M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, pages 54–62, 1999.

[16] A. Salden. Self-organizing e-business systems. In *TACC 2008, Budapest Tutorial and Workshop on Autonomic Communications and Component–ware, Budapest, Hungary*, July 2008.

[17] A. H. Salden and M. Kempen. Sustainable cybernetics systems – backbones of ambient intelligent environments. November 2004.

[18] J. Valk, J. P. Larsen, P. van Tooren, and A. ter Mors. Channel-based architecture for dynamically reconfigurable networks. In *BNAIC*, pages 246–253, 2005.

[19] I. C. Yu, E. B. Johnsen, and O. Owe. Type-safe runtime class upgrades in Creol. In R. Gorrieri and H. Wehrheim, editors, *Proc. 8th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'06)*, volume 4037 of *Lecture Notes in Computer Science*, pages 202–217. Springer-Verlag, June 2006.