A Pattern Language and Repository for Service Network Management

Ulrich Scholten, Nelly Schuster, Stefan Tai
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{firstname.lastname@kit.edu}

***Abstract*— Successful service platform operators foster their market performance by leveraging economic network effects, which implicitly control service ecosystems. Explicitly, third party services are used to complement the platform's intrinsic value to the users. Platform operators' key to success is the initiation of a snowballing interplay of consumers' preferences and a respective portfolio of service offerings. The Dynamic Network Notation DYNO supports modeling such service networks from a service management perspective, while defining system-interaction control and exploiting network effects. However, there is a need for an evolving and reusable base of experience that allows researchers and platform operators to learn from and to share knowledge on best practices. To this end, we introduce service network management patterns based on DYNO. In addition, to exploit the dispersed applications through various market segments, we present a community-driven pattern repository. The repository applies coordination and review means to ensure quality of patterns without restricting creativity during the pattern design process.**

Keywords: service network management patterns, repository, coordination, community-driven, network effects.

I. INTRODUCTION

Service platforms have become a vital part of IT industry's growth segments. Practically, there are lots of examples for successful platforms such as Salesforce and Netsuite. With the advancement of IT services into many specific segments, a plentitude of niche platforms is likely to start off, meeting market segment-specific demands; such as the sensor-platform Patchube.com or the e-Learning-as-a-Service platform SchandEdutech.com. Increasingly, value creating activities are delegated to ecosystems of service providers and customers, creating new forms of 'service networks', which require platform operators to shift focus towards a federation of capabilities [1]. Market surveys and experiments in the context of platform control show that service network management is one important factor of success of these platforms [2, 3].

In general, IT-service management is about managing the whole service life-cycle including the service strategy, service design, service transition, service operation and continual service improvement [4]. *Service network management* focuses on essential aspects of the platform design process in order to implement a platform-based service strategy. The platform architect has to account for the platform's surrounding business ecosystem (e.g., service providers, consumers, competitors) and implement suitable structures and control mechanisms in order to harness the platform's network effects.

A multitude of scientists works on models and architectures for service platforms [3] or socio-economic concepts behind platforms [5, 6]. However, none of them looks into patterns and best practices of successful and unsuccessful service network management. Therefore, we propose *service network management patterns* as mechanisms for documenting and communicating knowledge about successful and failed service network approaches. Patterns in general help to "identify, name and abstract common themes" [7] in design. By capturing the information and intent behind a design, we can make knowledge reusable. Patterns are condensed descriptions of the properties that succeeded in solving a specific problem. For example, we found three patterns for service deployment on a platform in a range of analyzed platform examples: a free deployment approach, a programming-model based deployment procedure and a programming-environment based deployment procedure.

Service network management patterns (a) provide a common vocabulary for platform architects and business analysts to communicate and document concepts as well as to explore management alternatives and (b) serve as reusable base of expertise through solutions retrieved from experience.

In order to aggregate experience from many unconnected communities (e.g., researchers from different disciplines, professionals from various market segments), we propose to capture patterns in an evolving *repository* of service network management patterns. Studies in the field of crowdsourcing in wikis have shown that coordination mechanisms like document structuring performed by few coordinators helps improving the quality of documents [6]. Addressing high quality requirements in the context of platform architecture modeling, we present coordination mechanisms, assuring quality of individual patterns as well as of the repository as a whole.

The contributions of this paper are twofold. First, we provide a pattern language to capture service network management patterns. The pattern language is based on the Dynamic Network Notation DYNO, a notation for modeling complex service network constellations [3]. Second, we bring together the pattern language and our model and tool for generic collaborative document creation [12]. In order to address the particular requirements of the pattern repository, we instantiate and adapt the collaboration model with specific coordination protocols and rules for collaborative contribution and refinement of service network management patterns.

The remainder of the paper is structured as follows: we start with the definition of service network management patterns and the introduction of a suitable notation (Section II), followed by a pattern example (Section III). Building on that, we present the collaboration model for the management and evolution of a service network management pattern repository, including suitable coordination protocols and rules (Section IV). Section V describes the demonstration of feasibility through a prototypical implementation, followed by a summary of related work (Section VI) and a conclusion and outlook (Section VII).

II. Service Network Management Patterns

A. Definition of Service Network Management Patterns

The definition of service network management patterns is built on terminology and concepts of pattern languages [8] and design patterns [7]. Patterns can be described as "abstract problem-solution pair, applicable for a specific environmental context" [9]. Plenty of pattern collections exist, addressing various contexts and levels of application. For instance, on a very technical level, the authors of [7] structure idiomatic class and object structures into design patterns, providing common vocabulary and constituting "a reusable base of experience for building reusable software". Workflow patterns [10] operate on a higher level of abstraction, describing conditions, examples, problems and solutions in workflow style expressions. Service network management patterns as introduced in this paper capture knowledge on best practices and experiences for controlling activities of participants as well as harnessing network effects in a service network platform ecosystem.

We differentiate between patterns, supplying best practice in specific applications (e.g., one-sided service platform with network effect, platform with programming model, platform with programming environment) and anti-patterns, practices where market studies proved that they lead to problems or underperformance (e.g., a Web service intermediary platform without base value contribution). Patterns might include other patterns.

B. Structured Documentation

Following [7], we suggest an initial structure for service network management patterns (Table I).

TABLE I. Structure of Service Network Management Patterns

Pattern Id	Unique pattern identifier.
Pattern Name	Meaningful name of the pattern.
Version	Version number.
Authors	The authors that contributed to the pattern, followed by the release version in brackets.
Status	Under revision, released.
Pattern Type	Pattern or anti-pattern.
Intent	Description of the addressed service network management problem.
Applicability	Description of the contexts where the pattern can be applied including preconditions.
Solution	Detailed description of the pattern, its accomplishment, limitations, etc.
Diagram	Graphical representation of the pattern.
Frequent Features	Detailed description of functionalities that are often, but not always applied.
Consequences	Description of pros, cons, and limitations.
Sources	If code for parts or all of a pattern can be downloaded at a URL, this URL is included here, accompanied by additional information e.g., license information and deployment guides.
Examples	Real life examples.
Included Patters	Cross reference to included patterns.
Related Patterns	Cross reference to closely related patterns.

The heading segments *pattern id, pattern name*, *version* and *authors* allow for distinct pattern identification. *Status* shows whether there is any limitation in usage due to revision or blocking. *Pattern type* indicates whether the depicted pattern is exemplary best practice or an illustration of solution that is likely to fail. *Intent* and *applicability* describe the addressed problem and the context. The detailed description of the pattern *solution* is followed by a *diagram* providing the graphical representation. Frequently used but not always applied features are listed in *frequent features*. Hereafter, *consequences* gives space for a discussion of advantages and disadvantages of the pattern. *Sources* allows displaying links to available code, *examples* describes real-life examples. The pattern structure closes with a list of *included* and *related patterns*. The template might be extended through the community.

C. Dynamic Network Notation DYNO for Terminology and Graphical Representation

The Dynamic Network Notation DYNO [11] is a domain specific notation, designed to support business analysts and system architects of service platforms in modeling complex network constellations of service provision and consumption.

The DYNO elements are described in Fig. 1. All modeling is done from the platform operator's perspective. Actors (e.g., internal solution supplier, external service provider or competitor) are modeled as *participants*. Unspecific groups of actors, e.g., the group of interested consumers or the group of potential service providers are specified as *participant groups*. DYNO allows modelers to specify different types of areas of influence: the *control area* delimits the controllable boundaries, e.g., the area within the corporate boundaries or a controlled area on an Infrastructure-as-a-Service environment. The control area can be segmented into *divisions*, for the sake of better clarity. The platform operator is limited to influencing what is outside the control area, the *influence area*. Examples are current or future customers or autonomous service providers who are interested to deploy services on the platform. The platform operator has no impact on those participants that are unwilling to cooperate or in the contrary want to disturb, e.g., competitors. Those participants are considered to be in the *noise area*.

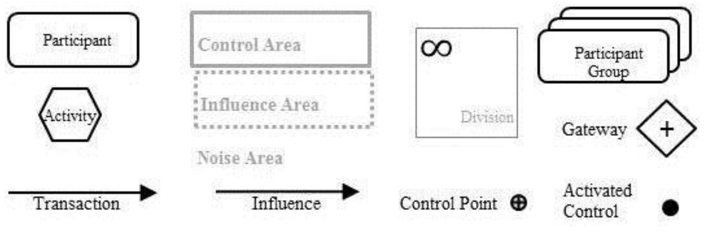


Figure 1. DYNO elements

(Inter-)actions take place within the control area and are modeled as *activities*. Examples for activities are service deployment, service consumption or platform subscription. Activities are provisioned through *transactions*, which are directed graphs, sourced from participants or other activities. Participants in the control area are provisioned from activities through transactions. Participants outside the control area are influenced through *influences*, sourced from activities.

All shapes located in the control area can be controlled through *control elements*. The control elements' configuration depends on their accommodating shape, the shape's connecting shapes and its location. Control elements consist of *control mechanisms*. Control mechanisms can be (a) prescriptive control of participants and activities inside the control area where the platform operator can prescribe the course of action, or (b) restrictive and sanctional control as enforcing mechanisms. Restrictive control only accepts inbound transactions, which are compliant to policy and regulations of the platform (e.g., programming model). Sanctional control incites an escalation routine at any time, if compliance of a respective activity to policy and regulations is not given anymore. (c) Motivational control groups all incentivizing activities towards participants within the control area. (d) Informative control gives the suppliers (customized) information on consumer preferences, requirements, etc. and thus supports the service provider towards an optimization of his services and leads in consequence to an optimized service portfolio. (e) Market-regulative control uses feedback from consumers to exert control.

Participants, participant groups or activities that have the potential to start off a first network effect should have a suitable *base value*. If a system has no base value, the architect should ask the question whether the platform has any potential to be successful.

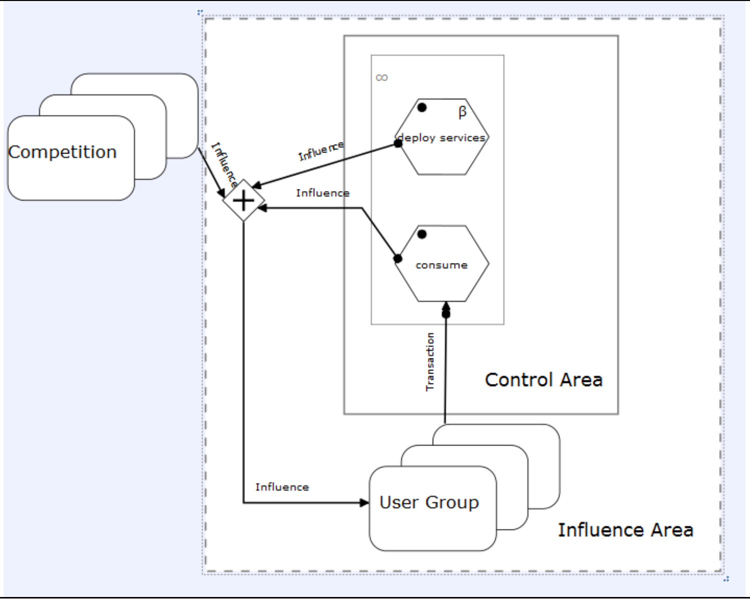
For a detailed description of all DYNO elements and the available control mechanisms, see [11].

III. Example for a Service Network Management Pattern

Table II gives an example for a service network management pattern. It describes a consumer-sided network effect in the context of a service platform.

TABLE II. THE CONSUMER-SIDED NETWORK EFFECT PATTERN

Pattern Id	GP00001.00
Pattern Name	The Consumer-Sided Network Effect Pattern
Version	1
Authors	Ulrich Scholten, Nelly Schuster
Status	Released
Pattern Type	Pattern
Intent	This pattern provides a solution for network effects that shall be exploited to grow the user base.
Applicability	The solution can be applied in service platforms as well as multisided markets. The precondition is a deployment environment, scalable enough to cater for potentially accomplished dynamic growth of service consumption through a consumer-sided network effect.
Solution	Members of a potential user group subscribe and *consume* services that were *deploy*ed by the platform operator. Regarding it as a dynamic system, the activity *consume* has the function of a stock. The more subscribed users consume, the more this stock is filled. The activity *deploy services* also acts as stock. This activity represents the base value, which initially sets of the system (indicated by the β-symbol). The quantity of users can – together with a quantity and quality of *deploy*ed services – motivate new potential users to subscribe to the platform. This motivation is weakened by competitive offers. Analyses of successful service platforms showed that they put explicit effort on positive user influence. They applied several control mechanisms: On the *influence*-edge, linking the *consume* activity and the *gateway*, platforms apply *informative* and *motivational control*. (a) *Informative control* amplifies the impact of the size of the user group. The nature of communicated information might vary; however platforms in all analyzed cases communicated the subscribed number of users. (b) Platform operators utilize *motivational control* in various shapes. The cross-section of all analyzed cases for the transaction edge pointing from the user group to the *subscribe*-activity was (c) application of *restrictive control*. I.e., the user has to subscribe to the services and to accept the platforms' terms and conditions. In the *consume*-activity, the platform exerts (d) *sanctional control*, i.e., he has the right and the power to exclude users. The deployment of services in the described pattern is an internal activity, therefore

	limited to (e) *prescriptive control* and (f) *sanctional control.* Exerting prescriptive control means to assign the task to produce and deploy a service within the platform operator's hierarchy. Sanctional control stands for the power to undeploy a service. The influence-edge, pointing from the *deploy* activity to the gateway includes the control mechanisms of (g) informative control. Informative control implies the clear and targeted information on the product. The activity *consume* needs to be placed in a scalable environment, to be able to respond to rapidly growing consumption. This behavior is realistic, as the activity is placed within a loop (= consumer-sided network effect).
Diagram	 Competition Influence Influence ∞ β deploy services Influence consume Transaction Control Area Influence User Group Influence Area
Frequent Features	• Additional to (b): Google+ incentivizes inscribed users to invite non-inscribed users through several procedures (e.g., upload of email addresses, self-paced invite of new users into circles). • In many cases the influence-edge, pointing from the deploy activity to the gateway also includes (h) motivational control and (i) market regulative control. Often suggestion mechanisms are used, e.g., 'other users, applying this service also applied XYZ'. Market regulative control describes the use of reputation mechanisms to reduce the entry barrier and to give decision support. • The activity deploy services can be filled by specific participants, representing the service design departments within the platform. Linking edges are *transactions*.
Consequences	The pattern is reduced to the core features of a service deployment. It blinds out any complex feature, e.g., replication or synchronization. Those need to be added on a context based approach.
Sources	not available
Examples	SAP By Design
Included Patterns	-
Related Patterns	The Two-Sided Network Effect Pattern, The Consumer-Sided Network Effect Anti-Pattern

The pattern in Table II could be turned into an anti-pattern simply by setting the base value in the activity *deploy services* to false.

IV. Collaborative Management of the Repository for Service Network Management Patterns

If there was a reusable base of knowledge on service management, platform architects could benefit from experience in other market segments. Pattern providers could in return benefit from the evolution of their suggested pattern. Such evolution might be grounded on technical progress (leading to modified patterns) or simply on more creative design. To reach as many market segments as possible, the knowledge base needs to be open to any possible user and to enable iterations of modifications. This openness however comprises the risk of low-quality contributions. The following collaborative but coordinated model for pattern repository management specifically resolves this dilemma.

The first requirement is to benefit from community-driven, intrinsically motivated value creation. The goal is to initiate and evolve individual patterns and the pattern language until they are generally accepted. The community members require a repository for collaboratively storing and managing patterns in the specified, but extensible, structure. The repository should support the use cases of creating, revising and approving patterns. As second requirement, the repository will need to enable coordination. For that, different roles need to be assigned during each pattern design process. *Contributors* deliver content according to their expertise, e.g., textual descriptions or diagrams, for a pattern. Existing patterns or parts thereof can be reviewed and updated by *revisers.* Any community member can play these roles. To assure quality of the patterns, the process requires *approvers* who state that a pattern is valuable and sufficiently described in its current version. An approval can be made at any time throughout the collaboration, by any community member. Throughout collaboration, dependencies between activities (e.g., writing and approval) as well as patterns and pattern parts need to be managed without restricting space for self-organization and creativity. To create a straightforward process the creator of an initial pattern should be the *coordinator* of this pattern. A coordinator can specify dependencies between activities or patterns and rules for managing them. For instance, he might assure that the author of a solution description is notified whenever an update to a diagram happens. The coordinator might delegate description tasks to certain community members. For instance, he might control contributions by assigning tasks (e.g., revisions, descriptions, approvals) to other members of the community, e.g., if an author does not get active during a defined period of time. To avoid deadlocks, the solution should envisage the backbone of a core team of revisers who can be asked to help out in case of non-response.

A. Collaborative Pattern Composition Model

To meet the first requirement, we propose to instantiate and adapt the service-oriented collaboration model introduced in [12]. Environments for open collaboration, e.g. wikis or open source repositories, enable community-driven content production. However, they do not envisage explicit approval processes or the assignment of responsibilities during collaboration. The service-oriented collaboration model promises to support the required balance of flexibility and coordination. According to this model, documents are dynamically composed of services provided by collaboration participants. We represent service network management patterns as documents which are composed of content delivery, revision and approval services. The repository contains an arbitrary number of such pattern documents. In the original collaboration model a single coordinator delegates work assisted through semi-automated dependency management. The envisaged repository, however, requires being open to unrequested contributions from the community as well as allow for a multitude of coordinators able to define and manage dependencies. How this is accomplished is shown in the following.

Fig. 2 presents the pattern composition model which describes how a service network management pattern is composed of services. In the course of collaboration, a coordinator (the initial creator of the pattern) decomposes a pattern into an ordered tree of (expected) *results*. Each result represents a part of the pattern, e.g., a diagram or a textual description of the solution. Results can be added to or removed from the pattern by the coordinator throughout the collaboration process. A result can also be a full pattern which includes text or diagrams.

Each result might be associated with an arbitrary number of *contributions*, which are activities for provisioning, revising, approving or publishing result contents. Important contributions for service network management patterns are the provisioning of parts of the pattern description (e.g., pattern Id, intent, solution) through contributors, the revision of these parts through revisers and the approval of a pattern document through approvers. With each contribution, a concrete *service* can be associated which is responsible for providing the contribution on request. These services can be provided by human beings (e.g., for pattern descriptions, revision remarks), software services (e.g., Id generator, DYNO editor for graphical representations), or the Web (e.g., examples, source code repository). This service-based approach allows pattern coordinators to easily integrate contents from external tools and services into patterns. In order to reuse patterns in other composed patterns, they are accessible as services on their own.

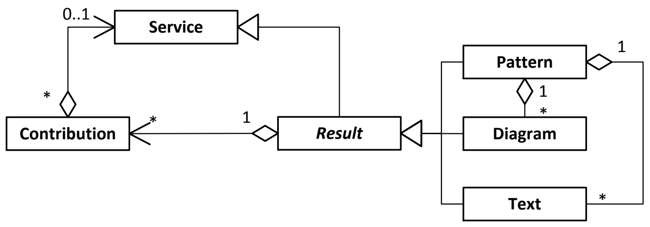


Figure 2. Composition model components and their relations, based on [12]

As an example, Fig. 3 shows parts of the consumer-sided network effect pattern from Section III as tree-based composition of results, contributions, and services. The pattern consists of an ID provided by a software service, the description of the pattern intent written by a human participant, examples on the Web, as well as a diagram modeled in DYNO. In addition, the pattern contains review and approve contributions.

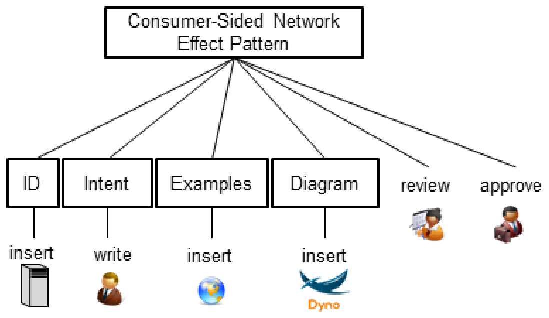


Figure 3. Example 'consumer-sided network effect' with results (boxes), contributions and services

As patterns should follow a certain structure with mandatory fields and required services (e.g., a pattern draft writing service, several approval services) as defined above, we suggest storing this structure in a pattern *template* in the repository. This template could be instantiated by a pattern coordinator at the time of creation of a new pattern in the repository.

B. Coordination Model

To meet the second requirement, the collaborative pattern composition model is complemented with an orchestration model consisting of two components. First, basic service *coordination protocols* control the binding of services to contributions as well as service calls. Second, *coordination rules* allow human coordinators to control the collaboration on an application level, i.e., manage dependencies between pattern parts and control execution of services. A coordination rule engine supports human coordinators through automating coordination protocols as well as executing the rules specified through the coordination rule model. In [12], we described protocols and rules for a hierarchical coordination mechanism where a human coordinator decides, who is allowed to participate in the collaboration. In this paper, we define additional protocols and rules and show how the collaboration model can be used to support a community-driven, more open style of coordination.

1) Coordination Protocols

The pattern gets interactive as services are called and deliver content as response to the calls. Before a service can be called, it needs to be associated to a concrete contribution. We defined *service binding protocols* for communication between coordinator and service provider. In the protocol shown in Fig. 4, a self-motivated community member who wants to provision a specific pattern section can ask the coordinator of this pattern for binding to the respective provisioning contribution. The coordinator can accept the request for binding or decline it. If the coordinator intends to invite specific service providers (e.g., for revision), the same protocol can be used with interchanged roles (coordinator-driven binding as presented in [12]).

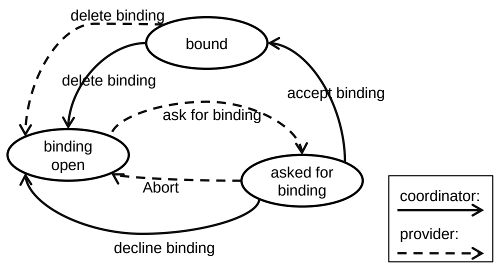
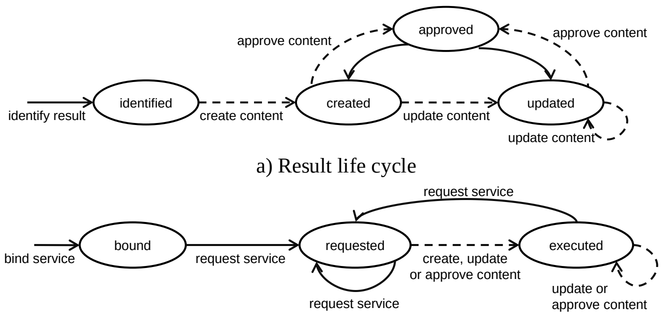


Figure 4. Service binding protocol. Ovals denote the state of the contribution; dashed arrows are decisions made by the service provider, solid arrows are decisions made by the coordinator. Starting state is *binding open*.

As soon as a service binding is established, a service can be requested which is represented in the *request/response protocol*. Fig. 5 shows the life cycles of a result and of a contribution.



b) Contribution life cycle

Figure 5. Service request/response protocol.

State transitions in the result life cycle are triggered by activities performed by an associated service. As an example, the result for the intent description is identified by the coordinator of the pattern as a required part of the pattern. A contribution for delivering the text for the intent description is appended and a service bound to this result, which is able to write the required content. Now, the service is *requested* to deliver the text. As soon as the service creates the intent, the result transforms to state *created.*

2) Coordination Rules

In order to avoid bottlenecks, manual effort of a coordinator needs to be limited. We use an event-condition-action (ECA) rule mechanism in the pattern repository to (a) automate service bindings and request/response protocols for all patterns in the repository and (b) manage dependencies between individual patterns and pattern parts. On each state transition in the protocols described above, events are emitted which are input to rules. These rules can trigger actions like service calls. Rules are specified by coordinators and can be modified during the collaboration. The rules in this section are specified in pseudo-code.

(a) Automating protocols: The protocols involve several coordinator activities which are automated by the coordination engine through the definition of default rules in the repository. In order to support automated binding of service providers to contributions, the following rule is specified:

```
ON BindingRequested
  AND BindingRequested.contribution.state
    != bound
DO acceptBinding(
  BindingRequested.service,
  BindingRequested.contribution)
```

Whenever a service provider asks for binding according to the service binding protocol, a `BindingRequested` event is emitted. For instance, a community member asks to be bound as author of the solution section of a pattern. On every occurrence of such an event (keyword `ON`), if the contribution is not already bound (keyword `AND` for conditions), the requested binding is automatically accepted by the rule (keyword `DO` defines the action), i.e., the requesting service is bound to the requested contribution.

As the `acceptBinding`-method also changes state of a contribution, an event is emitted denoting the new binding. Based on this `BindingAccepted` event, the service associated with the contribution is automatically called. For instance, the bound contributor is requested to write the solution section of the pattern.

```
ON BindingAccepted
DO requestService(
  BindingAccepted.service)
```

(b) Application-specific dependency management: The ECA rule mechanism allows coordinators to specify rules for specific patterns. In the following, we show various rules which we recommend a pattern coordinator to add to the rule engine for his pattern. In order to allow for comparable quality and design methodology of all patterns in a repository, a repository might provide a template instantiating these rules which can be used for the creation of a new pattern. This template should be combined with a template for structuring patterns. Still, a coordinator might add other rules. During collaboration, updates of parts can cause inconsistencies in other parts of a pattern. If, for instance, the diagram in the pattern is updated, the author of the solution section should proof whether the solution is still up to date, i.e., all services associated to the solution are called. This also applies the other way round. In order to implement this review cycle, we can specify the following rules. (Note: the terms in the quotes denote unique IDs of the elements.)

```
ON ContentCreatedOrUpdated(
  result=“diagram”)
DO requestService(“solutionRevision”)

ON ContentCreatedOrUpdated(
  result=“solution”)
DO requestService(“diagramRevision”)
```

Patterns can be logically composed of other patterns. As an example, the consumer-sided network effect pattern is part of the cross-sided network effect pattern (the composed pattern). Once a pattern is updated, which is part of a composed pattern (e.g., the consumer-sided network effect pattern is upgraded through a base value contribution, which results in updates of the diagram and the solution), the service providers of the composed pattern are called in order to check whether an update of the composed pattern is required as well.

```
ON ContentCreatedOrUpdated
AND
belongsTo(ContentCreatedOrUpdated.result,
  “consumer-sided network effect”)
DO requestServices(“cross-sided network
effect”)
```

In an initial setting, three approvers need to approve a pattern. This setting is assumption-based and subject to optimization in future works. Approvals are performed by service providers in order avoid bottle necks at the coordinator as well as enable community-based approval.

Until approved, the pattern state is *under revision*. Then its state can be set to *released* and it can automatically be published (e.g., to a Website). In order to implement this approval process, a rule listens on `ContentApproved` events on the pattern (e.g., the cross-sided network effect pattern in rule specified below). If three `ContentApproved` events are detected one after the other (-> operator) and no content update event was detected, the rule calls the publish service (contribution with Id publishX):

```
ON ContentApproved(result=“cross-sided
network effect”)
-> (ContentApproved(result=“cross-sided
network effect”)
  AND NOT ContentCreatedOrUpdated(result
      =“cross-sided  network effect”))
-> (ContentApproved(result=“cross-sided
network effect”)
  AND NOT ContentCreatedOrUpdated(result
    =“cross-sided  network effect”))
DO requestService(“publishX”)
```

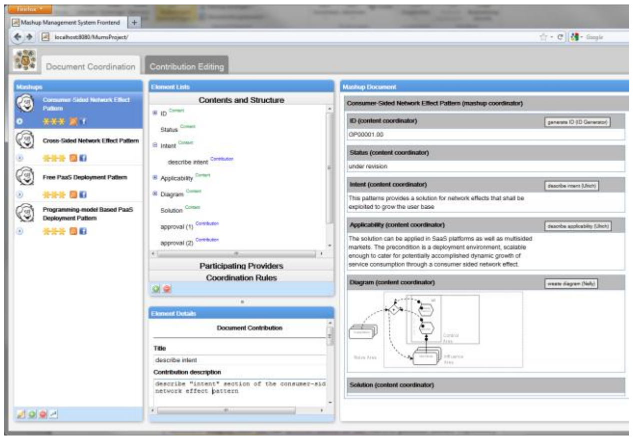
If a pattern is approved and an update is made to any of its results, the approval needs to be withdrawn, since it is based on a previous version of the pattern. This can be done through adding the following action to the rule two above:

```
DO withdrawApproval(“cross-sided network
effect”)
```

V. Implementation

In the previous section, we have described a community-driven approach for a repository of service network management patterns, including composition model, coordination protocols and rules. To determine the feasibility and utility of our design approach, we augmented our existing prototype tool and infrastructure [13] with the above described collaboration model, protocols and rules. The participating community can log into the frontend of the tool and use it as GUI for coordinating, composing, reviewing and approving service network management patterns. In the coordinator view, the frontend allows for *management of results* like solution and intent sections, *management of contributions* like reviews, *binding of service providers*, i.e. members of the community, and *dependency management* through defining and changing rules. The contribution editor allows community members to *perform services, i.e., deliver and update contributions*. For the design of diagrams for service network management patterns, we implemented and deployed a specific DYNO-editor [11] based on the open modeling platform ORYX [14] [1].

Fig. 6 shows a screenshot of the demonstrator in the coordination view depicting the cross-sided network effect pattern. The pattern is visualized as a tree structure as well as an HTML document, showing all sections as defined in Table I. Pattern diagrams are embedded as Scalable Vector Graphics (SVG). We have not yet implemented a service adapter for the DYNO editor but simply export the SVGs, produced by the DYNO editor. We further performed various settings of pattern management, e.g., delegation of contributions to community members or software services, provisioning of contributions through community members, update and successive update of sections.


Mashup Management System Frontend
localhost:8080/MumsProject/
Document Coordination
Contribution Editing
Mashups
Consumer-Sided Network Effect Pattern
Cross-Sided Network Effect Pattern
Free PaaS Deployment Pattern
Programming-model Based PaaS Deployment Pattern
Element Lists
Contents and Structure
ID
Status
Intent
describe intent
Applicability
Diagram
Solution
approval (1)
approval (2)
Participating Providers
Coordination Rules
Element Details
Document Contribution
Title
Contribution description
Mashup Document
Consumer-Sided Network Effect Pattern (mashup coordinator)
ID (content coordinator)
generate ID (ID Generator)
Status (content coordinator)
Intent (content coordinator)
describe intent (Ulrich)
Applicability (content coordinator)
describe applicability (Ulrich)
Diagram (content coordinator)
create diagram (Nelly)
Solution (content coordinator)
GP00001.00

under revision

This patterns provides a solution for network effects that shall be exploited to grow the user base

The solution can be applied in SaaS platforms as well as multisided markets. The precondition is a deployment environment, scalable enough to cater for potentially accomplished dynamic growth of service consumption through a consumer sided network effect.

describe intent

```
describe "intent" section of the consumer-sid
network effect pattern
```

Fig. 6: Coordination view of a pattern in the demonstrator showing the cross-sided network effect pattern

VI. Related Work

[15] suggest patterns of "mechanisms to govern and control the interaction among network participants". They do not consider any indirect relationships or control of unspecified participant groups as occurring in the context of network effects. To our best knowledge, there exists no work (yet) on service network management patterns in the context of cloud service engineering.

Many static online repositories for patterns in various contexts exist[2]. Also, the idea of exchanging ideas on patterns in a collaborative knowledge base is not new. For instance, for the Portland Pattern Repository (PPR)[3], a repository for patterns in software development, came with the first Wiki which allowed programmers to easily exchange and edit software design pattern ideas and information. However, the PPR does not come with structuring or approval functionality.

[16] describe an interactive pattern repository for inter-organizational business processes. This repository allows for a steadily growing pattern base. They define an extensive pattern meta-model, including classes for pattern descriptions including visualizations and relationships between patterns. In addition, they define roles and a pattern review process similar to ours where a certain number of reviews from the community are required to accept the pattern. However, community-driven collaborative authoring of patterns, partitioning of pattern descriptions, automated reviews or approvals based on update events, as well as the integration of external services, e.g., from software systems or the Web into pattern descriptions, are not envisioned. Several Web- or groupware-based approaches for collaborative management of architectural knowledge artifacts (including patterns) exist (e.g., [16, 17]) well suited for informed decision making during design. However, they do not allow for the integration of external sources (e.g., editors for graphical representations like DYNO editor, or sources from the Web), do not form a community-driven approval process or support flexible coordination of artifact dependencies.

VII. Outlook and Conclusion

In this paper we suggest a community-driven approach to build a repository for quality-assured service network management patterns. DYNO is presented as the vocabulary and graphical representation to describe service network management patterns. We propose a coordinated collaborative model and environment to author and evolve patterns. We expect added value and improved service and service platform design through shared knowledge made possible by this repository. Instantiation demonstrated feasibility. Growth of the community of participating experts to larger size is now required to allow for evaluating usability and applicability of the respective patterns and for the overall pattern repository approach. Only after the repository has attracted a statistically relevant size of users, it can serve as test-bed for alternate configurations of composition and coordination model.

[1] Test environments are publicly accessible under http://eorg-mosaic.aifb.kit.edu:8080/pattern-repository/ and http://www.DynoCloud.org.

[2] For instance, Workflow Patterns: http://www.workflowpatterns.com/; Service Interaction Patterns: http://math.ut.ee/~dumas/ServiceInteractionPatterns/ ; Enterprise Integration Patterns: http://www.eaipatterns.com/eaipatterns.html.

[3] Portland Pattern Repository, http://c2.com/ppr/

Potential configuration points are the number of required approvals, the number of coordinators as well as the possibility to delegate coordination responsibilities The patterns as suggested now (e.g., in our example) are still relatively basic due to the initial status of the patterns and the little iteration which our patterns went through. However, the community-driven approach promises to bring profoundness and completeness into the description.

As regards the service network management pattern repository, we plan to extend it into various directions. We plan to build a service adapter for the DYNO editor which allows community members to directly integrate and listen for updates of DYNO diagrams utilizing the adapter framework of our prototype which allows for building service adapters for 3rd party systems. Furthermore, we plan to evaluate the usefulness of the integration of external contents from the Web, e.g., source code repositories for pattern implementation code or examples. Traceability of pattern evolution and versioning of patterns is also an important issue which we plan to look at in future work. The proposed pattern repository focuses on the coordinated creation of patterns rather than the management of a large and growing number of patterns. Future work therefore needs to address information management issues like pattern duplicates, contradictory or inconsistent patterns.

From a broader perspective, the collaboration approach with rule-sets opened to coordinated, community-driven cooperation promises applicability in many contexts where crowd contribution is required but quality needs to be guaranteed.

References

[1] M. Cusumano and A. Gawer, "The Elements of Platform Leadership," MIT Sloan Management Review, vol. 43/3, pp. 51-58, 2002.

[2] S. Scholten, et al., "Composite Solutions for Consumer-Driven Supply Chains: How to Control the Service-enabling Ecosystem?," Proceedings of the 3rd academic symposium on Supply Management, Würzburg, Gabler-Verlag, 2010.

[3] U. Scholten, et al., "Perspectives for Web Service Intermediaries: How Influence on Quality Makes the Difference," E-Commerce and Web Technologies, pp. 145-156, 2009.

[4] E. Guldentops, et al. (2003). Board Briefing on IT-Governance [electronic book]. Available: http://www.isaca.org/Knowledge-Center/Research/Documents/BoardBriefing/26904_Board_Briefing_final.pdf

[5] S. Scholten and U. Scholten, "Platform-based innovation management: Directing external innovational efforts in complex self-organizing platform ecosystems," 2010, pp. 1-12.

[6] G. Parker and M. V. Alstyne, "Innovation, openness and platform control," presented at the Proceedings of the 11th ACM conference on Electronic commerce, Cambridge, Massachusetts, USA, 2010.

[7] E. Gamma, et al., "Design Patterns: Abstraction and Resuse of Object-oriented Designs," in ECOOP '93, 1993, pp. 406-431.

[8] C. Alexander, et al., A Pattern Language: Towns, Buildings, Constructions. New York, NY: Oxford University Press, 1977.

[9] T. Raffelsieper, et al., "Requirements for a Pattern Language for Event-driven Business Activity Monitoring," European Research Center for Information Systems (ERCIS), Universität Münster, Münster2011.

[10] W. M. P. van der Aalst, et al., "Workflow Patterns," Distributed and Parallel Databases, vol. 14, pp. 5-51, 2003.

[11] U. Scholten, et al., "DYNO: A Notation to Leverage Dynamic Network Effects in PaaS Ecosystems," in International Conference on Service Oriented Computing & Applications SOCA 2011, Irvine, 2011.

[12] N. Schuster, C. Zirpins, and U. Scholten, "How to Balance Flexibility and Coordination? Service-oriented Model and Architecture for Document-based Collaboration on the Web," Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on , vol., no., pp.1-9, 12-14 Dec. 2011.

[13] N. Schuster, R. Stein, and C. Zirpins, "A Mashup Tool for Collaborative Engineering of Service-Oriented Enterprise Documents," Information Systems Evolution, pp. 166-173, 2011.

[14] G. Decker, et al., "A Graphical Notation for Modeling Complex Events in Business Processes," in 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC), Annapolis, MD, 2007, p. 27.

[15] V. Kartseva, et al., "Control patterns in a health-care network," European Journal of Information Systems, vol. 19, pp. 320-343, 2010.

[16] A. Norta, et al., "A Pattern-Knowledge Base Supported Establishment of Inter-organizational Business ProcessesOn the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops." vol. 4277, R. Meersman, et al., Eds., ed: Springer Berlin / Heidelberg, 2006, pp. 834-843.

[17] R. Meersman and Z. Tari, "On the Move to Meaningful Internet Systems, 2002-DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002 Irvine, California, USA, October 30-November 1, 2002," Proceedings. Springer. ISBN, pp. 3-540, 2002.