Knowledge Management in Software Ecosystems: Software Artefacts as First-class Citizens

Dominik Seichter, Deepak Dhungana, Andreas Pleuss, Benedikt Hauptmann

{dominik.seichter, deepak.dhungana, andreas.pleuss, benedikt.hauptmann}@lero.ie

Lero - The Irish Software Engineering Research Centre
University of Limerick
Limerick, Ireland

ABSTRACT

Collaborative development of software products across organisational boundaries in software ecosystems adds new challenges to existing software engineering processes. We propose a new approach for handling the diverse software artefacts in ecosystems by adapting features from social network sites. We promote artefacts to first-class citizens in such networks and specify different types of relationships between artefacts and actors. This helps in detaching tacit knowledge from vendors, suppliers, developers and users of an ecosystem and fosters easier management of software artefacts. We discuss this by example scenarios and present a prototypic implementation.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*productivity*; K.6.3 [**Management of Computing and Information Systems**]: Software Management

General Terms

Management

Keywords

Ecosystems, Social Networks, Knowledge Management, Software artefacts

1. INTRODUCTION AND MOTIVATION

Software ecosystems have emerged as a new engineering challenge encompassing different facets of software engineering (like software reuse, extensible architecture, collaboration, community building, etc.). Practically, software ecosystems represent frameworks of software reuse, where different organisations, user-groups or companies contribute to a software product [13, 6]. On a high level, software ecosystems are groups of organisations or teams working together to create software products. However, in a more detailed level, there are a many tools, different artefacts like architecture, components or products themselves [5] that need to "work together", i.e., that need to be created and maintained using collaborative efforts that go beyond organisational boundaries.

Contrary to software development in a single organisation or with only a few external partners, centralised management is not feasible in a software ecosystem. Hence, novel methods for management are required, e.g., opening up the requirements engineering process, allowing the customers to vote on features, share road maps and coordinate release schedules [10]. Typical *product management decisions* that have to be dealt with in a software development project are related to the *release schedule*, *product road map*, *project goals*, *dependencies* [15] etc., and all of these involve shared knowledge. Of course, this is the case for any software development project, but dealing with the distributed and shared knowledge becomes a "killer criteria" for the success or failure of a product in software ecosystems, as any collaborating party makes decisions based on these pieces of information independently, which adds to the complexity of the problem of decisions making in a software ecosystem.

Currently, product management decisions are available only "internally" to the teams producing an artefact in a software ecosystem. This needs to be improved for two reasons: (i) whenever new actors (e.g., developers, suppliers or users) join an ecosystem, they may require information about the complex dependencies among shared artefacts, which can be difficult to get because it is beyond the borders of one organisation, and (ii) as actors can "leave" the ecosystem at any time, their leave is associated with loss of information. We therefore propose to explicitly build "social network sites" of software artefacts, which enables new actors to get the required information easily. Apart from that, our approach helps to reduce "knowledge drain", whenever actors leave an ecosystem.

We aim to improve the current state of practise by creating an infrastructure for the management of software ecosystem artefacts. We lift the status of shared artefacts in an ecosystem to "first-class citizens". We attach important information required to make product management decisions to the artefacts themselves to extract the tacit knowledge

ECSA 2010 August 23–26, 2010, Copenhagen, Denmark.

of involved stakeholders and make it thereby visible to the whole ecosystem.

Our contribution is a "community" of software artefacts, that are aware of the status of other related artefacts. Our approach is inspired by the success of social network sites [8]. A social network site is typically a Web 2.0 platform, where people can communicate and manage friendships and relations among each other. Many social network sites, e.g., Facebook[1], LinkedIn[2], Xing[3], etc., have proven to be well suited for communication between human beings. We extend this idea to create a network of software artefacts. Software artefacts like code, components, services, models or documents can interact with actors and other software artefacts. This emphasises on the importance of information associated to artefacts, rather than to the teams involved in creating them. We provide an infrastructure for software artefacts to interact with each other.

The rest of the paper is structured as follows. In the next Section we analyse the concepts and features of social network sites and how they are useful for managing contacts and relationships between friends or colleagues. Then, we present our approach for a social network site for software ecosystems where artefacts are first-class citizens, in Section 3. This approach is demonstrated on a prototypical social network for variability models in a software ecosystem. In Section 4 we show four example scenarios where social networks are useful in the context of software ecosystems. We present and overview of related work in Section 5 and conclude the paper in Section 6, where we also give an outlook on possible future research steps.

2. BACKGROUND: SOCIAL NETWORKS

A social network is a form of community, where interactions and communications of the actors are supported by the technical infrastructure [8]. We analyse the features provided by a social network site based on the example of Facebook, one of the most popular social networks. We identify key characteristics, which make Facebook particularly useful for fostering friendships and maintaining relationships.

Most social network sites have two concepts in common: (i) a profile for each user and (ii) relationships between users [14]. In this paper, we identify eight features that are important for managing "friends" and relationships in a social network site. These are mapped to or applied in the context of software artefacts later in the paper.

Profiles: A profile is used for presenting information about a participant of the social network site. The nature of this information is mostly static and does not change very often, e.g., name, contact addresses, or date of birth etc.

Wall: The wall is a part of the users profile, where users can publish their current status or comment on current activities, similar to online-services such as Twitter[4]. Additionally, other users can write on the wall of every other user, too, making it a feature for public communication.

News feeds: The news feed is a personal feature of every user of a social network site, which collects information from the walls of connected users and presents these pieces of information ranked by importance. Other users can in turn comment on the information presented by the news feed. This fosters immediate communication as friends change their status, others see the change in their news feeds and can react immediately on the changes.

Data sharing: An additional feature is publishing of data, like photos from the last vacation or writing blog entries about the last sport event. Again, this encourages other users to comment on the published data and/or rate it. Further data types used in social networks are videos, music and links, though any kind of file could be shared.

Teaming: For managing relationships and categorising them, social network sites support different ways of linking oneself to others. For example, direct connections can be made (e.g., *friends* in Facebook). These direct connections are notified about all status updates or newly published data. Secondly, it is possible to set up organisations of people in groups. These groups are usually more anonymous, but share several common interests. Everyone, who is part in a group is informed about information posted to the group.

Searching: In social networks, it is essential to be able to find the people one is interested in. One way to find new friends is to become friend with someone who is a friend of your friends. Another way is the usage of an integrated search platform. Hereby, it is notable that the search results are organised so that first friends of direct friends are displayed, then their directs friends and so on.

Suggestions: Most social network sites also have a suggestion system that recommends people or groups with similar interests. For example, one user joins a group of people with an interest in software ecosystems. Similarly, groups may be recommended by analysing interests of one's friends. Information in the news feed is usually also enriched with information from the suggestion system, i.e., information that might be of interest to a user is displayed more prominently in its news feed.

Messaging: Relationships between humans are mainly about communication. For this reason, a social network provides several means of communication. Rating and commenting on status updates or published data is a form of public communication that is visible to everyone or a specified group. Apart from that, private communication is an important feature that is provided using chats or other messaging systems or other means of synchronous or asynchronous communication.

Our approach promotes software artefacts to first-class citizens in a social network site and shows how this can deal with some of the issues related to knowledge management in software ecosystems.

[1]http://www.facebook.com
[2]http://www.linkedin.com
[3]http://www.xing.com
[4]http://www.twitter.com

3. APPROACH

We propose to extend the idea of social network sites to different artefacts, like code, components, services, models or documents, in use within the context of software ecosystems. Our approach is based on the assumption, that a network of tools and different artefacts is as important as the network of teams involved in software ecosystems.

3.1 Artefacts as first-class citizens

The stakeholders in a software ecosystem, – like software vendors, suppliers, developers, testers, users, etc. – communicate to each other through shared artefacts. These stakeholders enter and leave the ecosystem more often than the software artefacts do. By giving artefacts their own identity in a social network, we aim to make the network explicitly visible. The communication in a social network of artefacts is not focused on what the team members want to share with others, rather on what the artefacts "would want" to share with other artefacts.

In many cases, the interaction between the teams can be traced down to the interaction between the artefacts. For example, a typical team interaction in a software development project can be observed between developers and users, when a user reports a bug. Usually, the support team enters an issue into the bug tracking system, assigns it to a developer and informs other affected users about the new problem. Once the developer has fixed the bug, she reports it to the testers which try to validate the fix. Later, a new version can be deployed and the users can be informed that the problem was resolved. The artefacts involved in the above team interaction are the source code, the created issue, the tests, the developed product itself and any other artefact depending on the product. Reporting the issue creates a new artefact in the bug tracker, which gets also associated to the affected component or source code.

Furthermore, the information about the pending issue can be picked up by artefacts dependent on the product. Those could for example decide whether to use the product despite the issue or to automatically switch to an alternative if possible, which might be a different implementation or an older version. After the developer has committed a fix to the source code, the information from the commit log message is again associated to the product and, provided it is in a predefined format, it can be linked to the bug tracker issue, which can be closed automatically. Before closing the bug, automatic tests might be triggered to run and cause an automatic deployment of a new version. Consequently, the dependent artefacts are informed once more that the issue was resolved.

An example of a simplified social network with actors and artefacts and relationships between them is depicted in Figure 1. The *developers* responsible for the *WebService* are connected to it using an *owned By* relation. Other *3rd party developers* using the web-service have an interest in it and are connected to the web-service as well, to get status updates from it.

3.2 Social network of software artefacts

In a social network site, such as Facebook, basically only two kinds of relations exist between users: *friendship* and

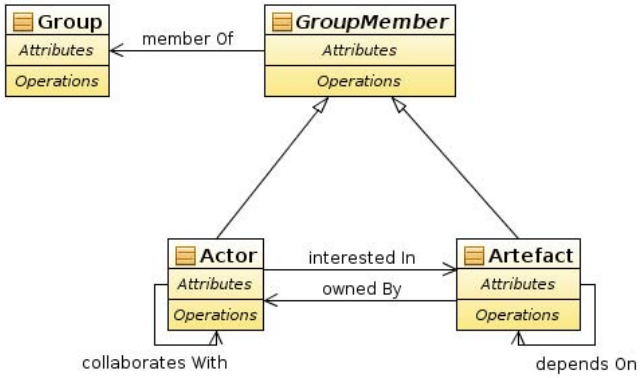


Figure 2: Conceptual model for a social network site, where artefacts are first-class citizens

group membership. We have to extend this simple model, to the conceptual model as shown in Figure 2. The conceptual model describes the entities of our ecosystem and the possible connections between them.

There are two first-class entities in the conceptual model, which we use as the meta-model of our social network site, namely Actor and Artefact. A future extension to this model might be, to introduce more specialised generalisations of these two entities to describe concrete actor or artefact types. Additionally, both actors and artefacts can be members of a Group.

There are different kinds of relationships possible between the key modelling elements. Actors can be connected to other actors (*collaborates With*), or can be members of a group. Artefacts can be related to their elements of the model in four different ways:,

owned By: Every artefact needs to have one or many owners, so that at least one person responsible for this artefact or with knowledge about the artefact is known at all times. This has been shown as a crucial need for developers [4]. In a normal team setting, team members are connected to other team members they work with. However, the responsibilities for artefacts change over the time. As the connection is made through artefacts in our approach, other actors or artefacts are always able to contact the owner of an artefact.

interested In: This relation describes an actor as being interested in a particular artefact, i.e., the actor wants to be informed whenever status updates are made to the artefact or new data is published by the artefact. Typically, if a development team uses an artefact, it is interested in it and wants all status updates from this artefact to be part of its news feed. This makes sure that information required to make decisions relevant to product management in an ecosystem is available to all interested parties.

depends On: In a large scale software development project, most artefacts depend on other artefacts. This relation has two important properties. First of all, it makes it possible to visualise the dependencies between artefacts in the ecosystem. Secondly, any artefact knows any other artefact that is

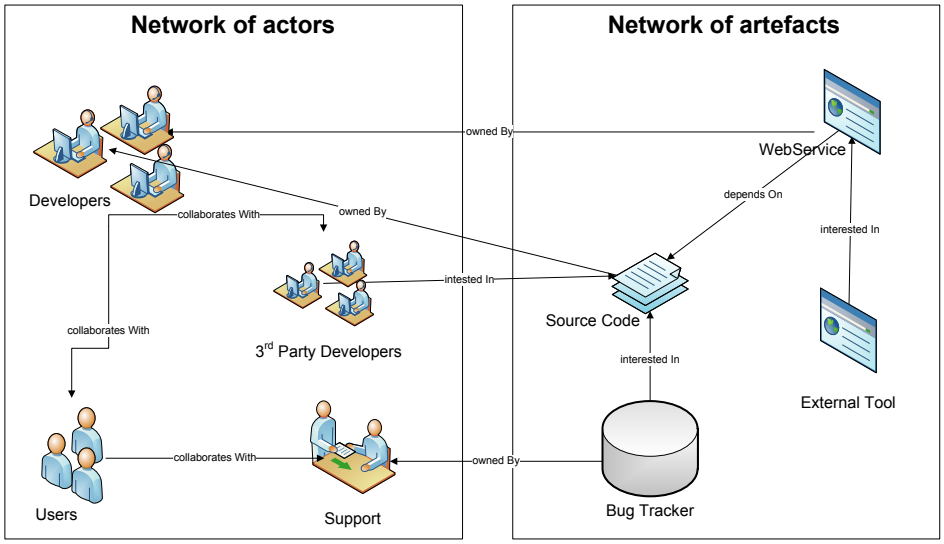


Figure 1: Social network consisting of actors and artefacts and their relations in the ecosystem

a dependency or a dependent. Therefore, important information from dependencies can be aggregated in news feeds or certain messages can be sent along these dependency relations.

member Of: Similar to existing social network sites, both actors and artefacts can be members of groups. An example for a group is to link all actors and artefacts involved in a certain product in the ecosystem together, among them developers, users, source code, models, services or, documentation.

3.3 Types of interactions

Now, that we have defined a meta-model of our social network site, where users and artefacts can co-exist, we have to define how these can interact. There are four possible interactions in such a social network.

Actor → Actor: Actors can interact with other actors, as it is the case in any other social network. This includes social networking features as messaging, data sharing, teaming or, influencing the suggestion system by rating others comments or published data.

Artefact → Actor: Promoting artefacts to first-class citizens allows artefacts to interact with humans as well. Any status change of the artefact is also visible in the news feed of all actors connected to the artefact. An artefact can create automatic status updates from sources such as source code repositories, bug trackers or static code analysis that are of interest to connected users. Begel and DeLine present this in their work [2] (see Section 5).

Actor → Artefact: As artefacts are first-class citizens, actors can interact with them in the same way as with any other actors on the social network site. This includes rating and commenting on status changes or published data. We distinguish between two types of interactions when artefacts are involved.

1. *Enriching with information:* Rating or commenting on information published by an artefact is defined as associating information to it. The actual content of the attached information does not matter for the artefact, but is only made available to other interested parties and can for example be used to react in an immediate discussion on a certain change of an artefact. For example, a team announces that an interface is going to be changed on the interface's wall. All related actors see the information in their news feed and comment on it or rate it.

2. *Interpreting messages:* Every artefact can provide several kinds of messages, which it understands by itself and can react onto them automatically in a reasonable manner, like triggering an action or starting a negotiation with some other artefact about a certain resource (e.g. allocate time in a test environment). These direct interactions are similar to interactions found between software agents and require a defined message or knowledge interchange format, e.g. KQML [9]. For example, the owners of an artefact can post information about a planned down-time on the wall and inform thereby dependent components, which can switch to alternatives or react on the down-time automatically.

Artefact → Artefact: This type of interaction occurs between artefacts. There are two categories based on whether information is interpreted automatically or an artefact is enriched with information. For example, information is interpreted automatically, if a component informs all dependents about an interface change, which is detected automatically after a commit, and the dependent components can automatically switch to an alternative, as it is shown in the example in Section 4.4.

3.4 Prototypic implementation

We have created a prototype (*iηvar*)[5] for a social network site of variability models in a software ecosystem. This prototype focuses, in its current implementation, on variability models as the only artefacts in the social network. Most interaction types surrounding actors are left out as they have been shown to be useful in the various popular implementations of social network sites.

Feature	Example
Profile	Profiles with information about users and variability models.
Wall	Comments can be attached to users and artefacts.
News feed	Provides information of new connections, new models and new comments.
Data sharing	Download of the actual variability model artefact.
Teaming	Composing of variability models and modelling their dependencies.
Searching	Searching for models and users.
Suggestions	Not implemented.
Messaging	Not implemented.

Table 1: Social network site features in the *iηvar* prototype.

Interaction Type	Example
Actor → Actor	Actors can communicate using wall comments.
	Sharing of variability models is another interaction of this type.
Actor → Artefact	An actor can select/deselect options in a variability model and create thereby a configuration.
	Actors can enrich artefact with information by posting wall comments.
Artefact → Actor	Not implemented.
Artefact → Artefact	Artefacts can be composed and can select/deselect options in other variability models in the same group using defined inter-model dependencies.

Table 2: Interaction types in the *iηvar* prototype.

The prototype offers the social network features of *profiles*, *teaming*, *walls*, *data sharing*, *news feeds*, and *searching* to an ecosystem of variability models. See Table 1 for a detailed description. Using *iηvar*, users can upload variability models to a central repository and create thereby a profile for a variability model. This profile includes information such as the owner of the model and the groups it is part of. *Teaming* is achieved by combining the models into groups and modelling their dependencies. Additionally, searching in all available models is possible.

Figure 3 shows a screenshot of the prototype, where several artefacts depend on each other. To make the social network features described in the paragraph above available, we had

[5]The prototype is available online at http://invar.lero.ie.

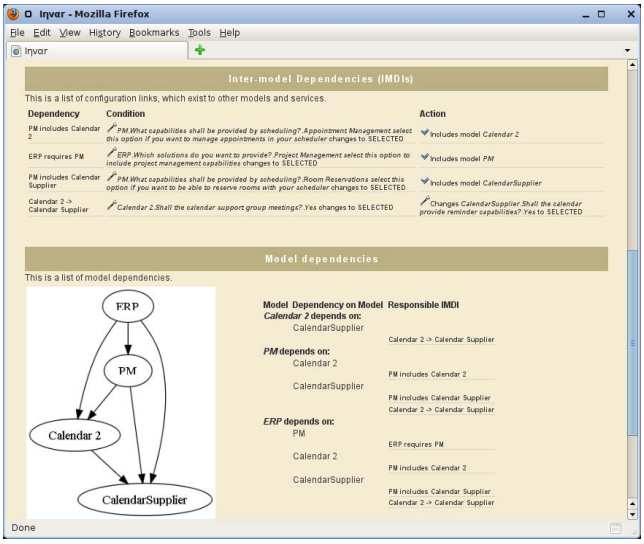


Figure 3: Screenshot of the *iηvar* prototype, showing the dependencies of four artefacts

to implement parts of our proposed conceptual model of a social network where artefacts are first-class citizens. We implemented the conceptual model relations *collaborates With*, *member Of*, *interested In*, *owned By* and *depends On* and the interaction types *Actor* → *Actor*, *Actor* → *Artefact* and *Artefact* → *Artefact* as part of the prototype.

Similar to other social network sites, different actors can be connected to each other, for example to follow updates of coworkers, using the *collaborates With* relation. The *owned By* relation is used to model ownership of artefacts. A variability model can be member of some or several groups, which implements the *member Of*-relation. So called *inter-model dependencies* are used to describe the *depends On*-relation. Additionally, the prototype supports the interaction type *Actor* → *Artefact* through the interactions of an actor when configuring a variability model or a group of variability models and by the possibility to enrich artefacts with information using wall comments. Artefacts interact with other artefacts (*Artefact* → *Artefact*) along the defined inter-model dependencies. A summary of implemented interaction types is listed in Table 2.

Every registered user in *iηvar* and every uploaded variability model has a profile page. These two types of profiles include information such as contact address, connections, ownership, and the wall, where comments can be posted.

Figure 4 depicts user profile in *iηvar*. At the top, (1) basic information of the user is displayed, like full name or contact information. Furthermore, three different kinds of connections are shown. The (2) connected users in the screenshot resemble the *collaborates With* relation of our concept model, (3) owned variability models are all those artefacts that are connected by an *owned By* relation and finally, the *interested In* relation is shown using a list of (4) variability models that the user is connected to. The wall can be found at the bottom of each profile, for users as well as for artefacts.

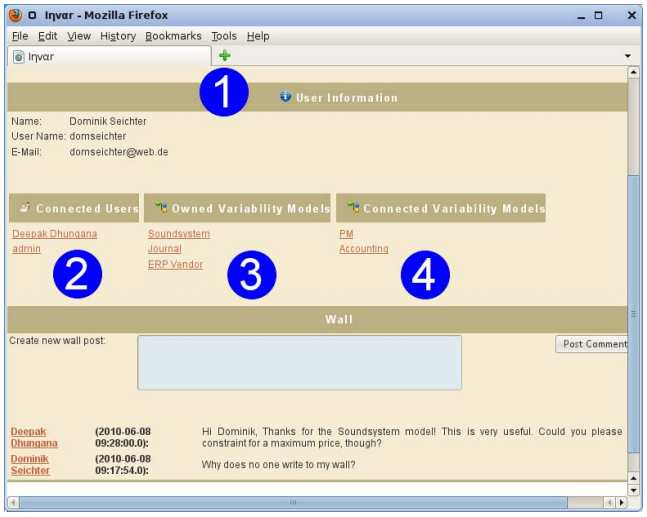


Figure 4: A user profile in *inνar*.

4. EXAMPLE SCENARIOS

We illustrate the benefits of adopting features from social network sites to software ecosystems using four possible scenarios. The examples were chosen to show any of the four interaction types that occur in a social network site with artefacts as first-class citizens. In each example, we assume that all actors and artefacts collaborating in a software ecosystem are connected using a social network site.

4.1 Change of an interface

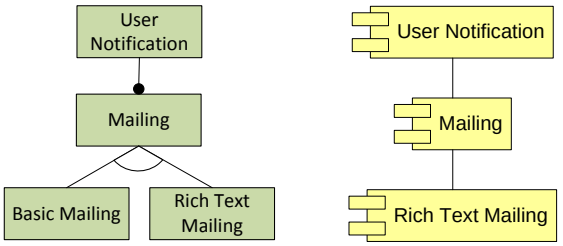
A team responsible for a popular artefact in the software ecosystem determines that the artefact's public interface needs to be modified to reach internal goals. Now, the team has to decide on the release schedule for the component. They announce their plans to change the interface in the next release on the wall of the artefact and add information about the release schedule. As the artefact is popular, the *Suggestions* system of the social network site displays information attached to the artefact very prominently in the *news feeds* of all its users. Some of the affected users comment immediately on the announcement to get into contact with the team and to find a consolidated release schedule or to suggest a different solution for the interface change.

In this scenario, an *Artefact* → *Actor* interaction fosters immediate reaction on a change and initiates communication between concerned teams.

4.2 Extension by a third-party developer

As software ecosystems are often used for a composition oriented development approach, it is a common scenario that third party developers want to extend their products with new features from existing components in the ecosystem. In particular, this can be fostered through the *Actor* → *Actor* interaction in our social network site.

For example, one third party developer aims to enhance her product with a new feature to win new customers. Now, the third party developer gets informed through the *news feed* of the social network site that another developer wrote about an interesting new feature in a software component.



(a) Variability Model (in FODA notation [12])

(b) Implementation

Figure 5: A small example ecosystem extract.

As both developers know and highly respect each other, the third party developer takes a look at the component. According to the profile of the component, it is used by many other users in the ecosystem and matches the third party developers needs. Therefore, the third party developer decides to connect to the component and uses it to enhance her own product.

In this scenario, the third party developer benefits from an *Actor* → *Actor* interaction which helps him to find a new component.

4.3 Change of responsibility

The responsibility for an artefact is transferred from an in-house development team to an off-shore team which takes over maintenance. An external user detects a problem in the component but does not know of this change in product ownership. Still, it makes no difference for the external user as the communication is via artefacts instead of human actors only. Rather than contacting the former responsible team the user attaches the information about the bug directly to the artefact. The information is automatically transferred to the issue tracker of the new maintenance team.

Here, the user benefits from the possibility of *Actor* → *Artefact* interactions.

4.4 Automatic dependency selection

Figure 5 shows an example ecosystem consisting of a variability model as shown in Figure 5(a) and three corresponding software components for its implementation (Figure 5(b)).

Let us suppose, the team developing the Rich Text Mailing component wants to release a new version, which requires a maintenance down-time. As the Rich Text Mailing component is part of a social network site the development team writes a message in a predefined knowledge interchange format to the wall of the Rich Text Mailing component and attaches thereby the information about the planned down-time. The User Notification component is connected in the social network to the Rich Text Mailing component as well as to the variability model. Internal reasoning of the User Notification component decides, that the proper reaction on a down-time message of a dependency is to request the variability model artefact for an alternative. The variability model suggests to use the Basic Mailing component instead. The User Notification component reconfigures itself to use the Basic Mailing component for the duration of the down-time.

This example shows the benefits of an *Artefact* → *Artefact* interaction in a social network site.

5. RELATED WORK

Some work has already been done to treat software entities as first-class citizens in their own social networks. For example, [3, 2] propose building social networks of code called Codebook. In Codebook, the interaction type *Artefact* → *Actor* is supported, i.e., artefacts publish automatic status notifications to their profiles, where the information comes from sources like code repositories or bug trackers.

Some researchers have also tried to mine repositories to harvest information required for building a social network [3, 4] and present as news feeds. Compared to their work, our approach has a broader focus on improving the collaboration between different teams over organisational borders and adds interactions with artefacts beyond retrieving information from them.

Alspaugh et al. [1] propose an automatic tool to check the compatibility of software licenses in an open architecture software ecosystem. We believe that such an automatic tool can also be implemented on top of a social network site for software ecosystems, where software licenses are artefacts that are connected to software artefacts with a new relationship type *licensed Under*.

6. CONCLUSIONS AND FUTURE WORK

Efficient management of shared knowledge in an ecosystem remains a big challenge because this is a relatively new field; researchers and practitioners have still to gain practical experience in dealing with software ecosystems; and there is a lack of proper tool support to deal with the multiple facets of the problem. For example, the adoption of software ecosystems increases the complexity in software development [7]. As software is composed of artefacts that are developed in different organisations or over organisational borders, the complexity increases naturally with every new artefact or team involved in the collaboration. An efficient mechanism for coping with the complexity is essential.

We believe that the complexity of software ecosystems can only be addressed by providing an efficient way for self-management to every actor in the ecosystem. Usually, no central management is possible in a software ecosystem, but a team developing an artefact in the ecosystem is able to make its own decisions (e.g. in the area of product management coping with release schedules, road maps and goals), if it has a suitable means to interface with the artefacts around it in the ecosystem. These means are provided by a social network site with artefacts added as first class citizens in our approach.

An added benefit of our approach is that it helps in the conservation of information. As information is now attached to artefacts instead of humans, it prevents a "knowledge drain" as team members change or leave.

An interesting point for further research can be taken from the *Research Agenda for Software Ecosystems* of Jansen et al. [11] in combination with our work: In what way do social network sites help to establish relationships with partners in a software ecosystem? We also encourage further research in the direction of what relationships are necessary in a social network site between artefacts, additionally to the very abstract relations *depends On*, *owned By* and *interested In*.

Several open problems remain, when introducing a social network site for software ecosystems. (i) Such a social network must be created right from the start when the software platform is opened as an ecosystem, as the social network can only unfold its full potential if all artefacts and all users are participating in the social network. Another problem is (ii) to find a suitable ontology and knowledge representation to allow for a useful communication between artefacts. Approaches from the software agent community could be used here. Although, we believe that a social network site is useful even without direct interaction between artefacts, we think that this could be an interesting point for future research.

Acknowledgements

This work was supported by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

7. REFERENCES

[1] T. A. Alspaugh, H. U. Asuncion, and W. Scacchi. The role of software licenses in open architecture ecosystems. In *First International Workshop on Software Ecosystems (IWSECO-2009)*, pages 4–18, Sept. 2009.

[2] A. Begel and R. DeLine. Codebook: Social networking over code. In *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 263–266, May 2009.

[3] A. Begel, K. Y. Phang, and T. Zimmermann. Codebook: Discovering and exploiting relationships in software repositories. In *Proceedings of the 32th International Conference on Software Engineering*, May 2010.

[4] A. Begel and T. Zimmermann. Keeping up with your friends: Function foo, library bar.dll, and work item 24. In *Proceedings of the First Workshop on Web 2.0 for Software Engineering*, May 2010.

[5] J. Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proceedings of the Second Conference Software Product Line Conference (SPLC2*, pages 257–271. Springer-Verlag, 2002.

[6] J. Bosch. From software product lines to software ecosystems. In *SPLC*, pages 111–119, 2009.

[7] J. Bosch and P. Bosch-Sijtsema. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, 83(1):67–76, 2010.

[8] D. M. Boyd and N. B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):article 11, October 2007.

[9] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *CIKM '94: Proceedings of the third international conference*

on Information and knowledge management, pages 456–463, New York, NY, USA, 1994. ACM.

[10] S. Jansen, S. Brinkkemper, and A. Finkelstein. Business network management as a survival strategy: A tale of two software ecosystems, 2009.

[11] S. Jansen, A. Finkelstein, and S. Brinkkemper. A sense of community: A research agenda for software ecosystems. In *ICSE Companion*, pages 187–190. IEEE, 2009.

[12] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. 1990.

[13] D. G. Messerschmitt and C. Szyperski. *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press, Cambridge, MA, USA, 2003.

[14] A. Richter and M. Koch. Functions of social networking services. In P. Hassanaly, A. Ramrajsingh, D. Randall, P. Salembier, and M. Tixier, editors, *Proc. Intl. Conf. on the Design of Cooperative Systems 2008*, pages 87–98, Carry-le-Rouet, France, May 2008. Institut d'Etudes Politiques d'Aix-en-Provence.

[15] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma. On the creation of a reference framework for software product management: Validation and tool support. In *IWSPM '06: Proceedings of the International Workshop on Software Product Management*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.