Understanding Software Ecosystems: A Strategic Modeling Approach

Eric Yu and Stephanie Deng
Faculty of Information, University of Toronto,
Toronto, Canada M5S 3G6

Abstract. Software ecosystems is an increasingly popular form of industry organization promoted by leading software vendors. Due to its novelty and complexity, many issues remain unclear for the various parties involved in the ecosystem. A transition from more conventional industry structures and relationships to an ecosystem will likely have profound impacts on business as well as technical design choices. In this paper, a preliminary attempt is made to use a strategic modeling approach based on the i* modeling framework to help understand software ecosystems. i* models are used to depict strategic dependencies between software vendor, third party developers, and end-users, and to help explore and reason about alternate ways for achieving strategic goals for each actor. We compare the buyer-supplier relationships in the traditional software supply chain to the open ecosystem format from a mobile platform vendor's perspective. Implications of the changing dynamics between the software vendor and its direct buyer and supplier are also discussed.

Keywords: Software ecosystem, strategic modeling, i*, industry structure, software supply chain, business model

1 Introduction

The software industry is constantly evolving and is currently undergoing rapid changes. Not only are products and technologies evolving quickly, many innovative companies are experimenting with new business models, leading occasionally to fundamental shifts in entire industry structures and how firms and customers interrelate [1]. Recently many companies have adopted the strategy of using a platform to attract a mass following of software developers as well as end-users, building entire "software ecosystems" around themselves, even as the business world and the research community are still attempting to get a better understanding of the phenomenon.

Software ecosystems (SECO) refers to the set of businesses and their interrelationships in a common software product or service market [2]. While the SECO approach appears attractive and is gaining momentum, its higher complexity brings tough challenges for its potential adopters [2]. For example, what factors need to be considered in order to succeed in a SECO? How can software vendors establish and sustain strong buyer-supplier relationships? Furthermore, the current lack of sufficient modeling techniques for SECO makes it harder for the participants and

potential adopters to conceptualize an ecosystem and therefore to establish viable strategies to address SECO challenges.

Some methods have been proposed to identify and analyze the complex relationships in software ecosystems. For instance, the Software Ecosystem Modeling (SEM) technique enumerates SECO objects to provide a holistic picture of the relationships in the software supply network [3]. However, these methods do not directly address strategic aspects of SECO suggested above. Other related work are discussed in a later section.

This paper illustrates the potential for using strategic modeling to help understand alternate SECO configurations. As a small example, we compare the buyer-supplier relationships in the traditional software supply chain to the open ecosystem format from a mobile platform vendor's perspective. We aim to reveal what is implied in such relationships and the impacts they can have on strategic organizational goals, from which different strategies in the buyer-supplier relationships may be illustrated. Hence, the understanding of key aspects around this issue can be improved.

Strategic modeling focuses on investigating and analyzing the strategic goals, intentions and relationships of each actor in a network of actors. It is a means to answer the "whys" behind an organization's decisions. Strategic modeling can be helpful in understanding organization issues at different levels [4].

The paper has the following structure. Section 2 provides a brief explanation of the i* modeling framework that supports strategic reasoning. Section 3 presents the modeling analysis and reasoning with two contrasting cases. Section 3.1 presents the buyer-supplier relations in the traditional software supply chain. Section 3.2 explains the buyer-supplier relations in the open ecosystem. The implications from the modeling analysis are discussed in Section 4. Section 5 concludes the paper with a reflection on the results.

2 A simplified example of buyer-supplier relationships

We use a small example to illustrate the basic concepts of i* strategic modeling, using general knowledge about the product-oriented software business domain.

Figure 1 shows a simplified set of relationships between a software vendor and an end-user. The overall operation of the software vendor is represented by the **task** *run software business*.[1] Running the software business consists of *getting software produced,* and *selling the software portfolio*. To get software produced, one can *build it in-house*, or one can *integrate software from an OEM supplier*. Selling software portfolio includes providing related services such as support and maintenance. For the purpose of this simple example, these model elements constitute a very rudimentary breakdown of the main components of running a software business. To model what it means to have a well-run business, we include model elements that reflect business performance objectives or criteria that would guide choice among strategic alternatives. For example, in choosing between building software in-house or integrating OEM software, one would consider factors such as *product quality* and *time-to-market* (TTM). These are factors that contribute to *high value* as perceived by

[1] Where appropriate, we use *italics* to highlight text that refer to model elements in the figures.

customers. High perceived value will help *attract new customers* and reinforce *customer loyalty*, both of which are essential for *building a large customer base*, which in turn contributes to *profitability*.

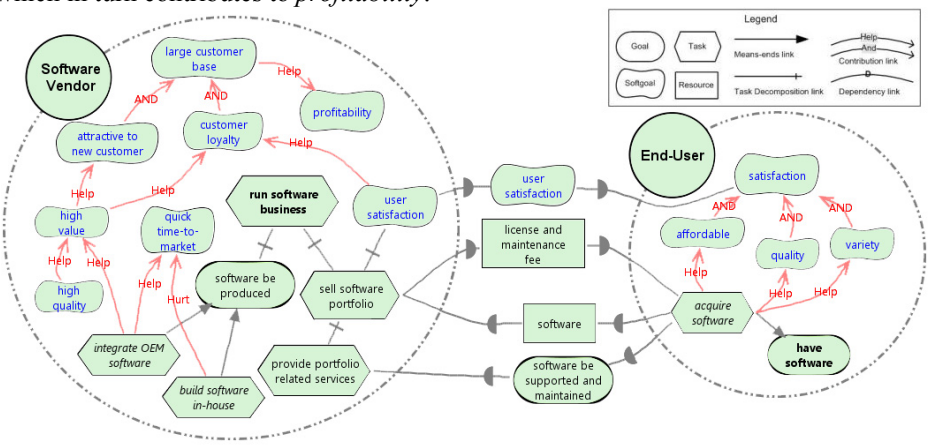

Fig. 1. A simplified i* model showing relationships between Software Vendor and End-User

In the i* strategic modeling approach, all actors are presumed to be acting strategically. The End-User, just like the Software Vendor, has different options and can choose those that are most strategically advantageous. In Figure 1, we show a much simplified situation in which the end-user acquires software from a vendor (as opposed to alternatives, not shown, such as building the software himself, or acquiring it from another vendor operating under a different business model, e.g., open source.) In doing so, he relies on the vendor to *support and maintain the software*. In return, the vendor depends on the end-user to pay *license and maintenance fee*, and to acknowledge *satisfaction*, e.g., by continuing to be a customer. For the internal strategic reasoning of the end-user, we show a very simplified situation where the user considers all of these three factors to be essential: good product *quality*, be *affordable*, and offering a *variety* of functionalities.

In an i* model, the main types of elements are goals, tasks, resources, and softgoals. These are called intentional elements because they represent what the actor wants to achieve. For examples, tasks are not simply process steps or activities (as in conventional process models), but are oriented towards some goals. In i*, a **goal** is a condition or state of affairs that the actor wants to achieve, without specifying how it is to be achieved. Modeling the condition (e.g., *software be produced*) as a goal indicates that there can be different ways (means) for achieving it. A **means-ends link** is used to connect a means (typically a task) to an end (typically a goal). A **task** specifies a particular way of doing something. It may be decomposed into its constituent elements through **task decomposition** links. The constituent elements may include subtasks as well as subgoals, resources, and softgoals. In Figure 1 the task *run software business* is decomposed into a (sub)goal *software be produced* and the subtask *sell software portfolio*. A **resource** is something that is needed for performing some task or for achieving some goal (and is hence also intentional). In Figure 1, *software* and *license fee* are examples of resources. A **softgoal** is similar to a

goal except that its criteria for satisfaction are not clear-cut and often not known a priori. Whereas ("hard") goals are either satisfied or not satisfied, softgoals are judged qualitatively and said to be sufficiently satisfied (satisficed) or partially satisfied. The **contribution link** is used to show how individual elements contribute to the fulfillment of a softgoal. For instance, performing the task *integrate OEM software* "helps" (i.e., positively contributes to) the softgoal *higher product value*, which in turn contributes to establishing *customer loyalty* and *attract new customers*. The "hurt" link indicates a negative contribution. The "make" and "break" links indicate that the contribution is sufficient to satisfice (or deny) the target softgoal. The help and hurt links indicate partial contribution, i.e., insufficient to satisfice (or deny) the target softgoal. Between actors, the **dependency link** indicates that one actor depends on another actor to have a goal achieved, a task performed, a resource to be furnished, or a softgoal to be satisfied. The different types of dependency imply different degrees of control and vulnerability of the depender [6]. In the example, the end-user depends on the vendor to provide the *software* (specified as a resource) while he receives license and maintenance fee in return. He also expects the *software to be supported and maintained* (a desired end result without specifying how it is to be achieved). If the end-user is not satisfied, then customer loyalty is not achieved, which means that the end-user is likely not to have future business with the software vendor.

i* is a modeling framework that provides explicit support to strategic reasoning through modeling actors' goals and rationales [6]. The Strategic Rationale (SR) model in the i* framework models an actor's intentional relationships both inside (as internal constructs) and outside (as external dependencies) the actor [5], and supports the process of exploring and evaluating alternative means to achieve specific goals of each actor. i* modeling can complement other types of SECO modeling by revealing the intentional and strategic dimension of relationships in SECO. Strategic reasoning from the viewpoint of each SECO participant is important for tackling the complexities of some SECO challenges, such as what strategies and tradeoffs are needed to build sustained buyer-supplier relationships. Since the focus of i* modeling is to reason about the strategic interests of various actors, i* models need not include every detail. Only those elements that would make a difference in assessing strategic goal achievement and in comparing alternatives need to be included.

The i* modeling framework was originally developed in the area of requirement engineering, to help analyze the social setting for which an information systems is intended [5]. It has also been applied to business process re-engineering, software process improvement, software architecture reasoning, security and privacy [16], and to reasoning about business models and disruptive innovations [7]. A version of i* has been adopted as part of the User Requirements Notation (URN), an ITU-T international standard [17].

3 Open Ecosystem versus Traditional Software Supply Chain

In this section, we apply i* strategic modeling analysis to compare the buyer-supplier relationships in two contrasting cases: the traditional software supply chain and the

open ecosystem, from a mobile platform vendor's perspective. We slightly extend the model introduced in section 2 to illustrate the reasoning for the two approaches. For example, we include additional important rationales of the software vendor, such as *increased solution gravity, vendor lock-in* and retaining *flexibility in product development*. While there are many other aspects in a software business, this study focuses mainly on the aspect of buyer-supplier relationships in producing and selling software products.

Since our intention is not to enumerate possible differences in the business models, for simplicity, we model the typical configuration in such settings and only include the essential actors. There are three actors in both models. For the traditional software supply chain (Figure 2), we consider *Software Vendor*, *End-User* and *OEM Supplier* as the main actors, while for the open system (Figure 3), the supplier is replaced by *Developer*. In transitioning towards an open ecosystem, the relationships between the software vendor and its buyers and suppliers evolve from a simple linear configuration to a more complex network of relationships. By employing the i* modeling technique, we are able to reveal the intentional elements implied in such relationships and their impacts on various actors through SR model constructs, from which two different strategies in the buyer-supplier relationships are illustrated.

The major literature sources we draw upon for constructing the models include the work done by Jansen, Finkelstein & Brinkkemper [2], Bosch [8], Popp and Meyer [9], and Popp [10]. The knowledge of mobile SECO is mostly based on publicly accessible information on the Internet. In the following subsections, we analyze the buyer-supplier relationships in the traditional software supply chain and the open ecosystem through i* SR modeling and analysis.

3.1 Adopting a Traditional Software Supply Chain Approach

Figure 2 illustrates the relationships between the actors involved in the traditional software supply chain. In a traditional software supply chain, there is a predominantly linear relationship from the end-user to the software vendor and from the vendor to its suppliers. Often the supplier is transparent or not directly visible to the end-user, although in some business models, the supplier does have a direct connection with the end-user [9]. For our illustration, we follow Popp and Meyer's characterization of the OEM supplier role [9] for the underlying business model between the software vendor and its supplier. Popp and Meyer also described motivations for supplier relationships from both the vendor's and the supplier's viewpoint. Many of these motivations have been reflected in the models (as softgoals).

For simplicity of exposition, we reuse most of the elements in the Strategic Rationale model for the Software Vendor as in Figure 1. We rename the main task as *run software business as supply chain*, with a means-ends link to the goal *software business be run*. The means-ends link indicates that the supply chain approach is only one way to run a software business. This will be contrasted in the next section with the open ecosystem approach. By having OEM suppliers, the software vendor can integrate complementary OEM software into its own solutions, which can help the vendor shorten time-to-market and gain higher customer value for its products. Higher value in turn contributes to attaining a large customer base. In Figure 2, OEM

supplier is a typical supplier in the traditional software supply chain. By *licensing software to the software vendor for its products* (main task inside the OEM supplier SR model), the supplier can *lower sales cost* and be a *niche player* in a specialized field. In order to serve multiple buyers, the supplier often prefers to keep its flexibility and stay independent of any particular vendor.

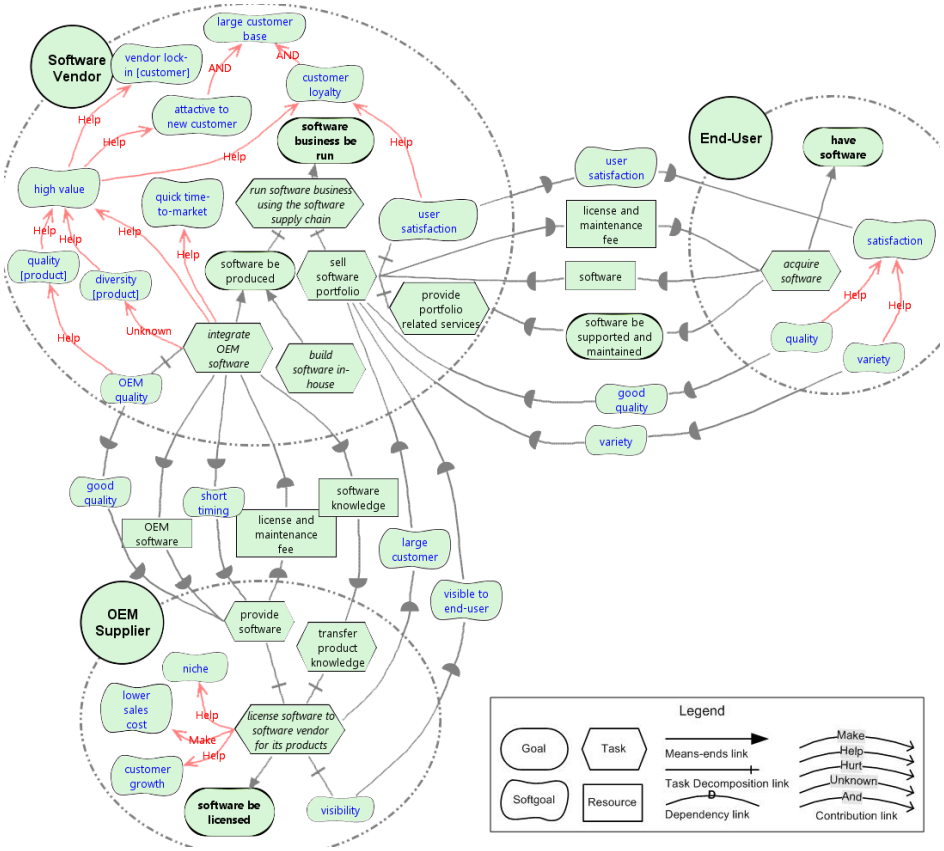

Fig. 2. Buyer-supplier relationships in the traditional software supply chain

The relationship between the software vendor and the OEM supplier can be analyzed in terms of the intentional dependencies that exist in both directions. The supplier provides *OEM software* to the vendor based on commercial licenses [9]. This is modeled as resource dependency *OEM software* from vendor to OEM, and the *license fees and maintenance fees* dependency in the opposite direction. Furthermore, the vendor depends on the supplier for the OEM software to be of *good quality* and to have *short release timing*. In the other direction, the supplier relies on the software vendor to reach *large customer base* which can help its *customer growth*. He also hopes to be *visible* to the vendor's customers to some extent through the vendor's distribution channels. In addition, the software vendor depends on OEM for *knowledge of the software* in order to do integration and to provide support.

End-User is the individual or organization who *acquires software* for private or internal use. It is the most common type of buyer in the software business world. The end-user needs to pay the *license and maintenance fee* for the *software* provided by the vendor. End-user depends on the vendor for *quality* and *variety* of the software, as well as for *software support and maintenance services*. The means-ends link indicates that *acquiring software* from the software vendor operating in a supply chain fashion is only one way to meet the goal of *having software*.

3.2 Adopting an Open Ecosystem Approach

A new strategy to engage buyer-supplier relationships for the OS platform vendor has emerged in the mobile SECO in recent years. In the traditional software supply chain, the software vendor provides both the platform and a range of applications to the end-user. In an open ecosystem, the software vendor provides a platform, but relies mainly on third party developers to create new value (through platform-specific applications) for the end-user, aiming to attract a large number of developers and end-users around the platform it provides.

Figure 3 depicts relationships among software vendor, developer, and end-user in a typical mobile platform ecosystem. The model is constructed using publicly accessible information on the Internet, and is not meant to accurately reflect any actual ecosystem.

Dependencies among the actors are more complex in an ecosystem than in the supply chain case. For an ecosystem to succeed, the software vendor needs to have a deep understanding of the rationales of the developer.

The software vendor provides *platform software* to the *developer* and the *end-user*, while depending on the developer to *develop creative applications for the platform*. In addition to *platform support and maintenance*, the vendor also provides the *market channel* for selling developer's applications. When the end-user acquires *applications* from the market channel, the vendor will have *revenue share* (in some form) on the purchase from the developer, while the developer is responsible for *supporting and maintaining the applications*. Moreover, since both are users of the platform, the software vendor needs to achieve both *developer satisfaction* and *end-user satisfaction*. Note that the software vendor can still integrate OEM components to its platform, rather than to build it all by itself.

In order to be *profitable*, the developer not only needs to use *the right platform* but also to create the right applications. *Adopting available open platform* from the software vendor is one efficient way for developer to monetize the ecosystem, but the platform should have the potential for *big market*, and is *easy to use* from the developer's point of view. While ease of use of the platform can minimize the effort by developers to contribute and therefore help to improve the developer's satisfaction, it is the big market that drives profitability and ultimately satisfies the developer.

Furthermore, the developer depends on the software vendor to provide a *powerful platform* and *fair condition for using* the platform. The developer also wants to be *visible* to the end-user through the market channel, allowing its applications to *get to market quickly*. The end-user now relies on the developer to provide *attractive*

applications that have *variety* and *good quality*, rather than relies on the software vendor for these.

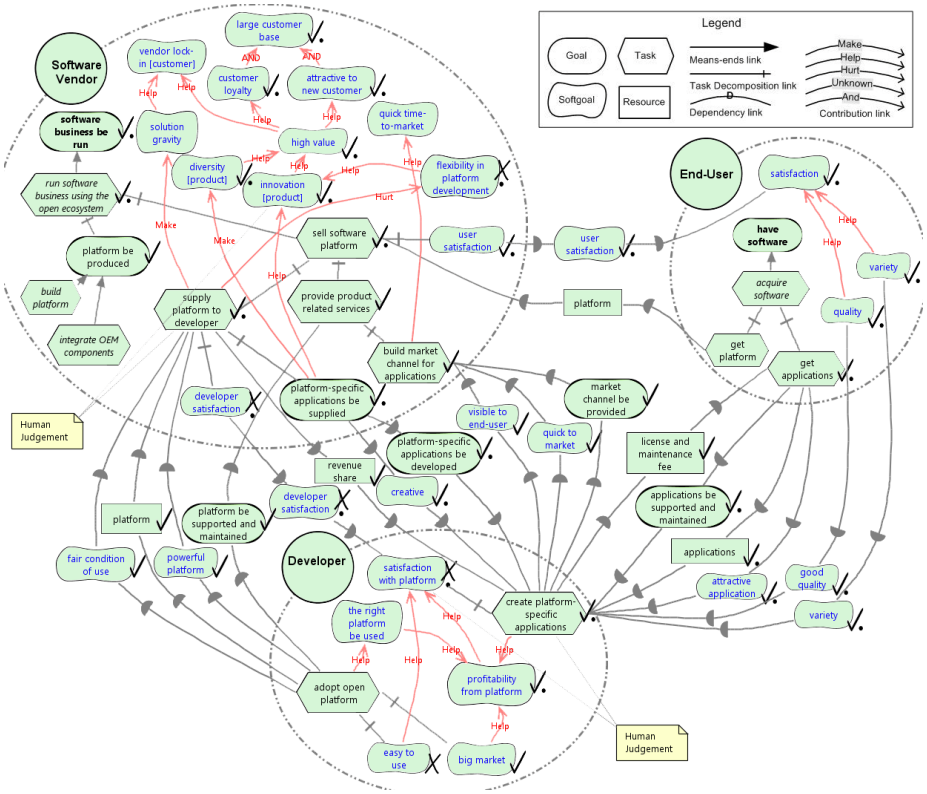

Fig. 3. Buyer-supplier relationships in the software open ecosystem

In adopting an open ecosystem strategy (Figure 3), we see that the software vendor emphasizes different business goals and chooses different ways to achieve them, compared to the traditional supply chain (Figure 2). Supplying platform to developer not only helps the software vendor increase *platform gravity*, which increases *customer lock-in*, but also achieves *diversity* in platform-specific applications and drives fast *innovation*. Diversity and innovation contribute to perceptions of *high value* by both existing and new customers. Similar to the traditional software supply chain, *time to market* is fast and satisfaction management contributes to attaining *large customer*. However, this approach does reduce the vendor's *flexibility* in modifying the platform once the platform is adopted on a large scale.

To more systematically assess whether desired goals are achieved, one can apply a semi-automated algorithm [11] to propagate qualitative evaluation labels across the graph, to help answer analysis questions from the models. Initial labels are set (if an element is satisfied or denied) and then propagated throughout the model using a combination of propagation rules and human judgment. The answers are interpreted from the propagated labels. Figure 3 illustrates the propagation of labels to answer the

question: if the vendor's platform is not easy-to-use from the developer's viewpoint, how it will affect the vendor and the end-user? Four types of labels are shown in the model: satisfied (***✓***), weakly satisfied (***✓.***), denied (***X***), weakly denied (***X.***). From the analysis, we can conclude that although the developer is not satisfied with the platform, the developer is still likely to use the platform if it is possible to make profit from the platform. However, the dissatisfaction will affect end-user satisfaction, thus impacting the software vendor's goal of achieving large customer base through the applications delivered. Note that at various points in the propagation, human judgment is required when the resulting value from a number of contributing values are underdetermined. Domain knowledge may be needed to judge the relative weights of different goals and to what degree an element is satisfied or denied.

4 Discussion

In the research agenda for SECO outlined in [2], the definition of SECO conveys the two central ideas that 1) a shared market in software products and services defines the boundary of a software ecosystem and 2) the technological platform or market acts as the common ground of the relationships among businesses within the ecosystem. It has been a long-established practice to build business partnerships (with customers, suppliers and partners), but the software ecosystem era brings new forms and interesting dynamics into this old practice. In SECO, as each individual player's success is tied to the outcome of the entire ecosystem [9], software vendors have been increasingly influenced by their external relationships. Without a good understanding of these relationships and their underlying rationales in terms of the strategic goals of each of the actors, a software vendor may incur unforeseen risks while embracing SECO enticed by its apparent benefits. Several observations can be made from the comparison between the traditional supply chain and the open ecosystem approaches.

First, the two models show that the buyer-supplier relationships have moved from a relatively simple linear chain to a more complex network of multilateral relationships. The complexity can be found in two ways: 1) increased number of dependencies, especially the associated softgoal dependencies, and 2) increased reciprocal dependencies among multiple actors.

Second, the role of supplier has evolved in certain ways. The OEM supplier in the traditional supply chain is purely a supplier of a needed software component, with no direct involvement with the software vendor's customers. In the open ecosystem, the developer is both a (platform) customer and an (application) supplier. In the supply chain, supplier is usually hidden behind the software vendor and has no influence over the end-user. In the open ecosystem case, while still having no control over the end-user, developer is largely visible and directly connected to the end-user.

Third, the model shows the possibility for the software vendor to exercise control via dependencies. It is important to note the difference between a goal dependency and a task dependency as conveyed in an i* model. A goal dependency indicates that the dependee has freedom on choosing how to achieve a certain desired state, while a task dependency means that the depender stipulates how the task is to be done. For example, in the open ecosystem, the software vendor has a goal dependency on the

developer for *platform-specific applications be developed.* By using means-ends links and subsequent decompositions on subtasks, different ways of how this goal may be achieved can be depicted as alternatives and strategies that either give more freedom to the developer or impose more control on the developer. In this way, different levels of control from the software vendor over the relationships can be expressed through the models and further analyzed.

Last but not least, while both are valid ways for a software vendor to run its software business, the two approaches have different impacts on the vendor's strategic goals (mainly appearing as softgoals) resulting from the separation of provision of the platform and its applications. When the software vendor provides both platform and application to the end-user, application *variety* is somehow limited because a single vendor does not have the capability to accomplish all the functionalities that the end-user(s) needs [8]. As the provisioning of them is separated, the open ecosystem approach offers better prospects for *diversity* and fast *innovation* that appear to be attractive to users. Moreover, this separation allows a larger set of users' needs to be met, which may be unfocused or neglected in the supply chain approach. It may also show the benefits in lowering the cost by supporting and maintaining a platform instead of a portfolio of products that are based on the platform.

Since the research community for SECO is quite new, there is only limited literature available on SECO modeling. A number of works have used modeling approaches to analyze software ecosystems and strategies of software vendors. Popp and Meyer [9] identified a set of roles that software companies can play in the ecosystem. They presented the relationships between different roles from a value model perspective, which focuses on value creation through the exchange of goods, services and payments. Recently Boucharas, Jansen and Brinkkemper [3] proposed the SEM technique to formalize modeling on SECO. The approach consists of two types of models: the Product Deployment Context (PDC) and the Software Supply Network (SSN). The PDC shows the software product architecture in the running environment while the SSN provides a holistic picture of the relationships between a vendor and its first-tier buyers and suppliers. Similar to Popp and Meyer's perspective, the enumeration of relationships between the actors in an SSN is based on value exchange (i.e., products, services, finance and content). In [12], [13], the interactions among ecosystems parties are modeled quantitatively using network analysis. Statistical techniques are applied to historical data. In [14], [15], mathematical models are used to study particular strategies (i.e., platform design moves, pricing strategies) of the software vendors. Statistical analysis or simulations are used to determine the impact of alternative strategies. In contrast, the strategic models in this paper aim to offer insights on the intentional and strategic aspects of the ecosystem actors and their relationships.

The i* method supports the expression and reasoning of the intentional aspects of goals and relations in the network of actors. When analyzing and designing SECO strategies, i* modeling can help situate the SECO specific goals (e.g., monetize on the ecosystem) with individual actor's goals, which provides an overall perspective on the important factors needed to be considered in an ecosystem. Furthermore, by elaborating means-ends at the goal level, i* models facilitate exploring alternatives (i.e., different strategies to achieve a goal). The evaluation mechanisms can assist

choosing among explored alternatives. Note that there can be goals at the strategic level (e.g., greater flexibility) and at the operational level (e.g., improved communication). Another benefit from the approach is that a body of knowledge can be accumulated through model building over many cases. This body of knowledge may then provide quality, ready-to-use information for future analysis of the SECO domain.

Limitations of the i* method include model scalability when the analysis involves a large number of actors, and reliance on the knowledge and experience of the modeler for achieving quality models.

The models we constructed in this preliminary study are based on knowledge gleaned from the literature. A more systematic method for knowledge acquisition and validation with domain experts will help improve the quality of the models.

5 Conclusion

In this paper, we provided an illustration of how strategic modeling using the i* framework can help in analyzing different configurations in the software industry. We used models to contrast how the dynamics of the buyer-supplier relationships have changed moving from the traditional software supply chain towards an open ecosystem. The i* strategic modeling method helps to improve the understanding around this issue in the following ways:

- it helps visualize the increased dynamics in the networked environment by making the relationships explicit, for both inside and outside the actors of interest, especially in revealing the intentional dimension of the relationships;
- it facilitates exploring strategies and alternatives to various degrees in the current setting; and
- it brings out a structure for systematic reasoning for how well the interests and concerns may be addressed by different configurations in the environment.

The i* models presented in this paper illustrate two strategies for the OS platform vendors. However, many variations are possible in each actor for further analysis. For instance, the variation can be related to strategies (e.g., the degree of control that a software vendor wants to impose on the external application development, to build or to exploit software). Variations can also arise from different business model designs (e.g., the types of services a software vendor provides, to leverage an OS-centric or an application-centric ecosystem).

The models in this paper are much simplified for the purpose of illustrating the modeling and analysis technique. For example, since we have focused only on comparing the product aspect of the buyer-supplier relationships, partners who mainly provide complementary services in the SECO have not been included in the models in the paper. In addition, empirical studies are needed to validate the approach, from which the effectiveness of strategy reasoning and design, using i* modeling, can be demonstrated through real world SECO case studies.

Acknowledgments. Financial support from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

References

1. Messerschmitt, D., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press, Cambridge (2003)
2. Jansen, S., Finkelstein, A., Brinkkemper, S.: A Sense of Community: A Research Agenda for Software Ecosystems. ICSE Companion 2009. pp. 187-190 (2009)
3. Boucharas, V., Jansen, S., Brinkkemper, S.: Formalizing Software Ecosystem Modeling. In: IWOCE '09 Proceedings of the 1st International Workshop on Open Component Ecosystems. ACM Press, New York (2009)
4. Yu, E.: Strategic Modeling for Enterprise Integration. In: Proceedings of the 14th World Congress of International Federation of Automatic Control (IFAC'99). Elsevier Science, Permagon. pp. 127-132 (1999)
5. Yu, E., Giorgini, P., Maiden, N., Mylopoulos, J. (eds): Social Modeling for Requirements Engineering. MIT Press, Cambridge (2011)
6. Yu, E.: Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. In: Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97). IEEE Computer Society, Washington, DC. pp. 226-235 (1997)
7. Samavi, R., Yu, E., Topaloglou, T.: Strategic Reasoning about Business Models: A Conceptual Modeling Approach. Information Systems and e-Business Management. 7(2), pp. 171-198 (2009)
8. Bosch, J.: From Software Product Lines to Software Ecosystems. In: SPLC '09 Proceedings of the 13th International Software Product Line Conference. pp. 111-119 (2009)
9. Popp, K., Meyer. R.: Profit from Software Ecosystems. Books on demand, Synomic GmbH (2010)
10. Popp, K.: Definition of supplier relationships in software, http://www.drkarlpopp.com/resources/ICSOBSubmission2.pdf
11. Horkoff, J., Yu, E.: Interactive Analysis of Agent-Goal Models in Enterprise Modeling. International Journal of Information System Modeling and Design. IGI-Global, 1(4), pp. 1-23 (2010)
12. Kabbedijk, J., Jansen, S.: Steering Insight: An exploration of the Ruby Software Ecosystem. In: Proceedings of the Second International Conference on Software Business. (2011)
13. Iyer, B., Lee, C. -H., Venkatramen, N., Vesset, D.: Monitoring Platform Emergence: Guidelines from Software Networks. Communications of the Association for Information Systems. 19 (1) (2007)
14. Liu, X., Lee, C. -H., Iyer, B.: The Impact of Design Moves on Platform Adoption: The Case of Microsoft Windows OS. In: HICSS '06 Proceedings of the 39th Annual Hawaii International Conference on System Sciences. IEEE Computer Society, Washington, DC. (2006)
15. Buxmann, P.: Network Effects on Standard Software Markets: A Simulation Model to examine Pricing Strategies. Standardization and Innovation in Information Technology, 2001 2nd IEEE Conference. pp. 229-240 (2001)
16. Yu, E.: Social Modeling and i*. In: Conceptual Modeling: Foundations and Applications. A. Borgida et al (eds). LNCS vol. 5600. Springer, pp. 99-121. (2009)
17. ITU-T, Recommendation Z.151 (11/08), User Requirements Notation (URN) – Language definition. (2008). http://www.itu.int/rec/T-REC-Z.151/en