# Using BPEL Process Descriptions for Building up Strategic Models of Inter-Organizational Networks

Dominik Schmitz[1], Gerhard Lakemeyer[1], Günter Gans[1], and Matthias Jarke[12]

[1] RWTH Aachen, Informatik V, Ahornstr. 55, 52056 Aachen, Germany
[2] Fraunhofer FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany
{schmitz,lakemeyer,gans,jarke}@cs.rwth-aachen.de

**Abstract.** In previous work, we proposed the prototype environment SNet for the representation and dynamic evaluation of agent-based designs for inter-organizational networks. A key feature of SNet is the automatic translation of extended i* models into the action language ConGolog. An issue we have not yet considered is how to arrive at the foundational and hopefully realistic i* model. Currently there is no support to incorporate information from already existing descriptions of business processes in an enterprise. BPEL is expected to play an important role in the future by enabling interoperability of different partners' business processes – not only in the web service domain. Once standardized a wide-spread availability of BPEL-based process descriptions can be expected. In this paper we suggest how to map BPEL descriptions into i* descriptions, thus opening the door to generating SNet simulations of business processes from BPEL descriptions.

## 1 Introduction

In previous work, we proposed the prototype environment SNet to model strategic inter-organizational networks, which are comprised of human, organizational, and technological actors [1]. A crucial aspect of these networks are the interdependencies among the various actors, which result, for example, from the need to delegate certain activities, which in turn requires a certain level of trust between the (human) members of the network. The agent-based graphical modeling language i* [2], which was developed for early requirements engineering, has proven to be particularly suitable as a modeling means in this context, because it explicitly deals with dependency relations, besides other notions like actors, goals, resources, and tasks. To capture the dynamic aspects of agent networks we proposed to amalgamate i* and the action formalism ConGolog [3]. To bridge the gap between the two formalisms we extended i* by features to describe task preconditions and effects. These extended i* diagrams are automatically translated into executable ConGolog programs, supported by the metadata manager ConceptBase [4]. Running simulations for different scenarios within a network is useful for analyzing its properties and can provide the foundation of a decision-support tool for network members.

In recent work [5] we introduced a decision-theoretic planning component for each network representative to run even more realistic simulations and improved the modeling facilities among other things by introducing a role concept [6]. The main current research effort concentrates on a closer contact with the real world: in [6] we considered already a real world entrepreneurship network. In this paper, we investigate now the possibility to use BPEL (Business Process Execution Language for Web services) [7] process descriptions as a starting point for building up the detailed i* model we need for analyzing strategic inter-organizational networks. The consideration of BPEL process descriptions results from the insight that it is very unlikely that a modeler will start the modeling of an inter-organizational network and the processes therein from scratch. He will base his work on existing process models and more general all experiences he has had so far.

Although BPEL is not undisputed (see [8],[9] for competing efforts) it must be considered as the most promising candidate for describing business processes – esp. in an inter-organizational setting – in a standardized way. Additionally there are already suggestions to combine agent technology and web services to finally realize the promise of a virtual enterprise [10]. Once the standardization effort is successful and if the trend towards web services continues this leads to the expectation that in the near future many business software products will be able to produce BPEL descriptions as an output. Thus providing an interface to BPEL is a natural step to support building up realistic i* models of an inter-organizational setting and the first step to incorporate already existing information on business processes in an enterprise in a systematic way. In this paper, we suggest how to map BPEL descriptions into the i* model. The goal is to eventually turn this into a completely formal mapping which can be automated.

The rest of the paper is organized as follows. In Sect. 2 we introduce our SNet simulation and modeling tool. In the following section (Sect. 3) we provide a short introduction into the usage of BPEL. The main contribution is then presented in Sect. 4 where we describe how and which parts of BPEL process descriptions can be mapped to extended i*. After shortly discussing some related work Sect. 5 ends the paper with a conclusion and a brief outlook on future work.

## 2 The Modeling and Simulation Environment SNet

### 2.1 The Architecture of the SNet Tool

We base our modeling and simulation environment SNet for inter-organizational networks on a combination of two formalisms: *i** – a graphical modeling language originally intended for describing early requirements – for statically modeling the network and *ConGolog* – a logic-based high-level programming language – for simulations so that dynamic aspects such as trust can be analyzed. We take an agent-oriented view in that each actor of an inter-organizational network is represented by a deliberative agent. We will discuss the features of the two formalisms in more detail later on. First we give a short overview of their overall interplay. The SNet architecture is depicted in Fig. 1.

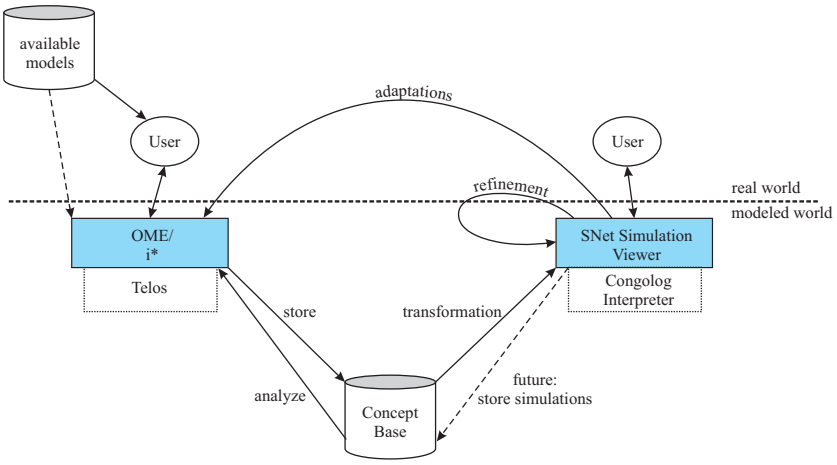


**Fig. 1.** SNet Architecture

We use *OME3* (Organization Modeling Environment) – a graphical model editor developed at the University of Toronto [11] – to build up static models of inter-organizational networks in the modeling language i* [2]. The semantics of i* is defined in the knowledge representation language Telos [12] which is also the formalism underlying *ConceptBase* [4], a deductive metadata repository. The ConceptBase query-language can be used for static analyses and especially for the transformation into ConGolog. The execution of the resulting ConGolog program is shown in a step by step view by the simulation viewer, which also provides access to control the simulation run, i. e. the user creates scenarios by simulating the pro-activity of network members and investigates how this and resulting delegations affect relationships (especially trust). Conclusions derived by the user from such simulations might lead to modifications of the model or scenario conditions which provide the basis for new simulation runs.

### 2.2 An Extended Version of i*

The i* framework is a graphical language and includes the *strategic dependency (SD)* model for describing the network of relationships among actors and the *strategic rationale (SR)* model, which, roughly, describes the internal structure of an actor in terms of tasks, goals, resources, etc. Compared to Yu's original formulation we added a few new features to SR models such as task preconditions. Figure 2 shows part of an extended SR model from the entrepreneurship domain with a focus on the roles *Venture Capitalist*, *Entrepreneur*, and *Faculty Member*.

The venture capitalist's task *choose_promising_entrepreneur* is decomposed into three subtasks, which are ordered using *sequence links* (an easy to use form of a task precondition). The task *suggest_business_idea* is delegated to the actor *Entrepreneur*. Goals like *ask_evaluation* provide a means to specify alternatives from which the instantiated agent (respectively represented network member)

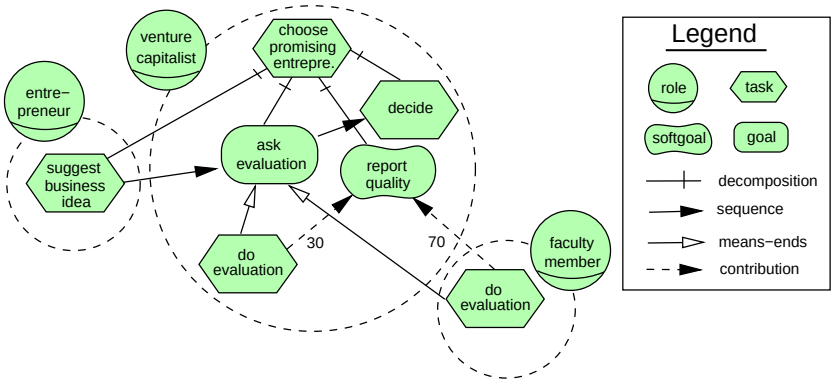


**Fig. 2.** Modeling in i*/SNet

can choose at run-time. In this example the venture capitalist can choose to do the evaluation himself or delegate this to a *Faculty Member*. Softgoals are used to specify the criteria on which the deliberative component bases its decision, e. g. *report_quality*. Tasks can have preconditions and effects, represented by their own model element denoted as a triangle (but not occurring in the example above), and produce or consume resources.

### 2.3 Mapping the i* Model to a ConGolog Program

ConGolog based on the situation calculus [13] is a language for specifying complex actions (high-level plans). For this purpose constructs like sequence, procedure, *if-then-else*, but also non-deterministic and concurrency constructs are provided. ConGolog comes equipped with an interpreter which maps these plans into sequences of atomic actions assuming a description of the initial state of the world, action precondition axioms, and successor state axioms for each fluent. For details see [3].

To enable simulations the abstract roles defined in the i* model have to be instantiated. The instances of a role (agents) differ in details as for example duration of tasks and softgoal contributions. We have to omit the details of the mapping here but simply mention that in addition to the ConGolog interpreter we provide an environment which equips each agent with a decision theoretic planning component to reason about which alternative (or partner) to choose according to the specified criteria (softgoals). See [5] for details.

## 3 BPEL

BPEL [7] the Business Process Execution Language for managing interoperability of business processes is currently under evaluation at the standardizing committee Oasis. It builds upon the web service stack (including SOAP, WSDL, etc.). BPEL's main idea is that while WSDL allows for the functional description

of web services, this is not enough if it comes to supporting business processes. Besides the interface of each web service the orchestration of services (requiring some stateful knowledge) has to be enabled. Accordingly BPEL provides the means to combine several web services into a new web service with all the necessary considerations of states and information exchange while avoiding to explicitly refer to instances of web services within this description. A suitable instance is supposed to be chosen at run-time (e. g. with the help of UDDI).

In the following we provide a brief introduction into BPEL by considering a small example. Due to limited space we cannot cope with all BPEL constructs. See Sect. 4.3 for some considerations about omitted parts.

**The Loan Service Example.** In the example the approval of a loan is provided as a web service. A customer enters some personal information and the amount of money he needs, the web service evaluates the request with the help of other web services, and eventually approves or rejects the request. To fulfill its task the loan approval service can make use of two other services: one for simple risk assessment and one for a more detailed approval analysis. Whether and which of these services are invoked for a given request at run-time depends on the request itself and exactly this is described in the business process description. So if the requested amount is higher than $ 10.000 the approval service is contacted. If it is below or equal to $ 10.000 first the simple risk assessment service is fed. If this service assumes a low risk the loan approval service immediately approves the request whereas in case a medium or high risk level is identified the approval service is contacted again.[3]

```
 1:<process name="loanApprovalProcess" ...>
 2:  <partnerLinks>
 3:    <partnerLink name="customer" myRole="loanService"
 4:                 partnerLinkType="loanPartnerLinkType"/>
 5:    <partnerLink name="approver" partnerRole="approver"
 6:                 partnerLinkType="loanApprovalLinkType"/>
 7:    <partnerLink name="assessor" partnerRole="assessor"
 8:                 partnerLinkType="riskAssessmentLinkType"/>
 9:  </partnerLinks>
10:  <variables>...</variables>
11:  <faultHandlers>...</faultHandlers>
12:
13:  <flow>
14:    <links><link name="receive-to-assess"/>
15:           ...omitted...</links>
16:    <receive partnerLink="customer" portType="loanServicePT"
17:             operation="request" variable="request"
18:             createInstance="yes">
```

[3] BPEL assumes a specification of the appropriate services (port type and operation) together with a suitable set of `<message>`s to be defined in WSDL.

```
19:      <source linkName="receive-to-assess"
20:              transitionCondition="amount=<10000"/>
21:      <source linkName="receive-to-approval"
22:              transitionCondition="amount>10000"/>
23:    </receive>
24:    <invoke partnerLink="assessor" portType="riskAssessmentPT"
25:            operation="check" inputVariable="request"
26:            outputVariable="risk">
27:      <target linkName="receive-to-assess"/>
28:      <source linkName="assess-to-setMessage"
29:              transitionCondition="risk-level=low"/>
30:      <source linkName="assess-to-approval"
31:              transitionCondition="risk-level!=low"/>
32:    </invoke>
33:    <assign>
34:      <target linkName="assess-to-setMessage"/>
35:      <source linkName="setMessage-to-reply"/>
36:      <copy><from expression="'yes'"/>
37:            <to variable="approval" part="accept"/></copy>
38:    </assign>
39:    <invoke partnerLink="approver" ...>
40:      <target linkName="receive-to-approval"/>
41:      <target linkName="assess-to-approval"/>
42:      <source linkName="approval-to-reply"/>
43:    </invoke>
44:    <reply partnerLink="customer" portType="loanServicePT"
45:           operation="request" variable="approval">
46:      <target linkName="setMessage-to-reply"/>
47:      <target linkName="approval-to-reply"/>
48:    </reply>
49:  </flow>
50:</process>
```

**Fig. 3.** BPEL Description of Loan Service

Separately from the above process description BPEL allows for the specification of `<partnerLinkType>`s by specifying at maximum two roles (minimum 1 role) which are involved in a partnership. A `<role>` refers to the port type of a web service. If there is only one role specified this means that there is no constraint placed on the other partner. In the example the following `<partnerLinkType>`s exist: *loanPartnerLinkType*, *loanApprovalLinkType*, and *riskAssessmentLinkType*. The details of the first are given below:

```
<partnerLinkType name="loanPartnerLinkType">
  <role name="loanService"><portType name="loanServicePT"/></role>
</partnerLinkType>
```

**Fig. 4.** `<partnerLinkType>` Definition

In the process definition these `<partnerLinkType>`s are instantiated (see Fig. 3, lines 2–9). But with instantiation it is not meant here to assign a dedicated service but to specify which of the roles is played by the described business process. In our example this is the role *loanService* concerning the `<partnerLinkType>` *loanPartnerLinkType* whereas concerning the two other `<partnerLinkType>`s the business process is only client and the partners play the only specified roles *approver* resp. *assessor*.

**Providing and Invoking Web Services.** Concerning communication i. e. web service interaction the three most important activities are `<invoke>`, `<receive>`, and `<reply>`. `<invoke>` is used to start a request to another service (e. g. Fig. 3, lines 24–26). BPEL allows for synchronous as well as asynchronous requests. A synchronous invocation is blocking, i. e. the process does not proceed any further until the requested service has provided its answer. In the example it is assumed that all services run fast enough to let synchronous requests be sufficient. In case an asynchronous request needs to be modeled the reply is gathered by a callback interface on which the process waits via a `<receive>` statement. The `<receive>` construct is also used for the provision of a service, i. e. a process description usually starts with a `<receive>` construct (Fig. 3, lines 16–18).[4] The `<reply>` construct is needed to answer a synchronous request (Fig. 3, lines 44–45).

**Flow Control.** A key feature of BPEL is the possibility to describe how the different invocations of services and answers to requests are timely related to each other via so called *structured activities*. Activities grouped together in a `<sequence>` construct are executed one after another. In contrast to this all activities combined in a `<flow>` (see example) are in general executed in parallel. But it is possible to specify arbitrary sequential relationships between these activities via `<link>`s. Every activity can be source or target of such a link and conditions can be associated with the sources which enable or disable the corresponding links accordingly.[5] The results of the link semantics as specified in Fig. 3 will become clear when the mapping of the example process is considered (see Sect. 4.2).

For simplicity and due to reasons of space we concentrate here on the constructs occurring in the example. BPEL provides some more constructs like `<switch>`, `<while>`, another communication construct (`<pick>`) as well as event, fault, and compensation handlers where the latter two belong to BPEL's sophisticated means to specify fault behavior.

[4] The setting of the `createInstance` attribute reflects that for each request a new process instance is created.

[5] The *transitionCondition* in the example is simplified since normally this should be a XPath1.0 expression referring to variables, message properties etc.

## 4 Mapping BPEL to Extended i*

Since BPEL claims to be sufficiently detailed so that an engine able to execute the BPEL process description can provide the described service as a new combined service, it has to cope with many nasty details which are not relevant to a more strategic investigation we are interested in. For example only message parts which have strategic effects esp. on the quality of a service must be regarded. Thus in the following we want to analyze which aspects of a BPEL process description provides valuable information for creating i* models.

### 4.1 Deriving Actors from BPEL Descriptions

The most high-level elements in i* are actors specialized into agents, roles and positions. Since agents refer to individuals they correspond to an instance of a service and thus are not relevant here neither on the i* modeling level nor in the BPEL descriptions. In contrast to this the abstract definition of a web service (via WSDL as a port type with an operation) can be mapped to an i* role since it comprises a self-contained functionality. For the naming we suggest to refer to the `<role>` specification within a `<partnerLinkType>` (see Fig. 4). Thus for the example we arrive at the roles *loan service*, *assessor*, and *approver*.

Additionally the `<partnerLinkType>` definitions describe relationships between different roles on an SD model like level without detailing the type of dependency (task, resource, goal, softgoal). While in general a task dependency will be most suitable it is up to the modeler to semantically analyze how much freedom is left to the delegatee and thus choose an appropriate dependency type. For the example we assume a task dependency to be adequate for all relationships. Altogether we arrive at the situation pictured in Fig. 5. Note that we added the role of a *customer* not explicitly mentioned as the partner role for a *loanPartnerLinkType*.

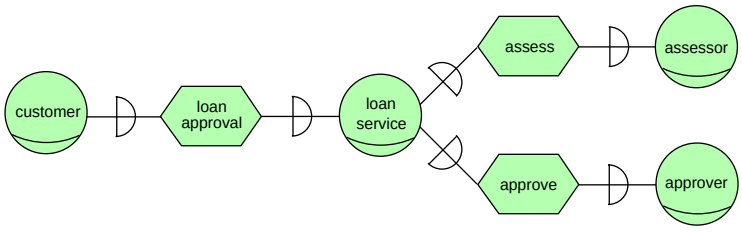


**Fig. 5.** Strategic Dependency Diagram

Introducing a role for each web service might be too detailed, but a later merging is not excluded so its a good starting point. Especially in case of repeated communication between business partners a continuous business process on both sides must be assumed which must be reflected in the model (re- and back-delegations). Another type of grouping which also exists in BPEL does not occur in the example but can also nicely be mapped to i*. Using the `<partners>`

element several <partnerLink>s (recall in a <process> <partnerLinkType>s are instantiated) can be grouped together to put an additional constraint on the web service chosen at run-time in that it must fulfill all the partner roles in the specified <partnerLink>s. This is much in the same spirit as i* combines several roles into a position if they are commonly played together.

### 4.2 Mapping the Process Body

After we have got now our basic actors available, we can focus on detailing their internals. Since a BPEL document describes only the details of one of the business partners involved we must in general expect several of such detailed descriptions to be combined into one i* model. Naming conflicts are not expected since we assume an i* model to be built out of the subjective view of one of the business partners. Accordingly the knowledge about the other actors is limited since they will not provide him with all the details about their internals. Instead he himself has to revise the models of their behavior according to his knowledge and experiences so far. Thus he has got to a large extent control over the naming and can avoid naming conflicts.

The body of the process has to be mapped onto the specification of the internal behavior of a role. To subsume all the activities we first have to generate a top level task element whose name can originate from the `name` attribute of the <process> element. For our example we have chosen *loan approval*.

**Mapping of Basic Activities.** Concerning inter-service communication only the <invoke> activity automatically results in an i* task element because due to the higher modeling level the receipt of a message and the provision of a reply need not be considered in i* models separately. Thus the starting <receive> activity is not mapped to an SR model element, but instead strategically relevant message parts are mapped to parameters of the top level task element. Accordingly the corresponding <reply> activity is also not mapped to an SR model element. But since SNet currently does not support return values for delegated tasks[6] if the return value is needed for the process description it must be mapped to a precondition/effect element which then can again be used as a precondition for other tasks.

The <invoke> activity is mapped to a task, but following its semantics this task is not associated with the current role, but with the one of the invoked service. Thus this is a delegation in the SNet naming and it is graphically represented by a decomposition link which crosses role boundaries. If – as suggested – a more detailed process description for this business process at the business partner is given this is used to detail it. If such a description does not exist the resulting top level task element is simply primitive and the naming must be derived from the associated `operation` attribute.

[6] It can only be decided whether the delegation was successful or not and possibly its contributions on accompanying softgoals.

The effects (and sequential relationships) of a `<receive>` activity corresponding to an asynchronous request must be associated with the task element resulting from the transformation of the matching `<invoke>` activity. Notice that SNet innately assumes asynchronous i. e. non-blocking communication. To emulate synchronous communication a suitable set of sequence links needs to be added.

The means to specify internal local activities in BPEL is limited – BPEL focuses virtually exclusively on orchestration. Only the `<assign>` activity which sets the value of a variable (see Fig. 3, lines 33–38) can be seen as some internal activity. But whether it should be mapped to its one primitive task depends on whether it has any strategic impact. In the example the local decision to approve a loan if the amount is below $ 10.000 and risk is low is mapped onto a primitive task. A name for this task can be derived from the destination variable possibly in connection with the names of links arriving at or leaving this activity. In the example we simply suggest *setMessage*.

**Mapping of Structured Activities.** Equally important to the transformation of basic activities is their timely relation to each other as specified using the `<sequence>` and the `<flow>` construct together with `<link>`s. A `<sequence>` relationship between activities can simply be reflected on the side of i* via usage of sequence links which have exactly the same meaning. Attention must only be paid if there are activities which do not have an equivalent on the side of i* (see above `<receive>`, `<reply>`). In this case the destination of the sequence links must be adjusted accordingly.

The default semantics of the `<flow>` construct is also the default of the combination of sub tasks (or goals) on the side of i*, i. e. everything is executed in parallel. Unconditioned links can again be mapped onto sequence links whereas conditioned links should be mapped to the more general precondition/effect element. This means the source activity has got some effect and the described condition is checked as a precondition for the target activity. In fact the modeler should make use of the separate specification of conditions to avoid doubling conditions as in the example: the check whether the amount exceeds $ 10.000 or not should be represented only as one precondition/effect element and a negated precondition link can be used as the precondition for the invocation of the simple risk assess service.

The above description does not yet reflect one property which makes the `<link>` semantics different from the `<sequence>` semantics. This is if the `joinCondition` of an activity (which per default combines incoming `<link>`s via OR) is not fulfilled the activity is simply skipped.[7] This corresponds to an 'if needed' style of task decomposition (denoted by dashed decomposition links) we already proposed in earlier versions (OR-task decomposition).

[7] In fact we assume here, that the `suppressJoinFailure` attribute of `<process>` is set to `yes`. If this is not the case a lot of fault behavior invocation has to be regarded which we omit here (see Sect. 4.3).

The result of the mapping of the loan service description to an SR model is given in Fig. 6.

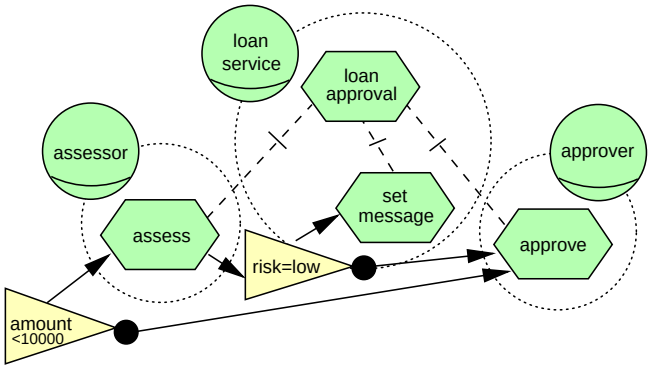


**Fig. 6.** Strategic Rationale Diagram Resulting from Transformation

It is important to mention, that it is not a necessity that every basic activity within a process description becomes a direct sub task of the top level task. Indeed it is much nicer to preserve the groupings resulting from the usage of structured activities like `<sequence>` and `<flow>`. In case a name is given (via the corresponding default attribute) it can be used to name this complex sub task. Otherwise we suggest to generate a dummy name (e. g. *flow_23*) and leave it to the modeler to choose a more suitable one later on.

### 4.3 Preliminary Evaluation of the Suitability of BPEL

As the previous presentation already suggests, BPEL is to some extent too detailed (e. g. explicit `<receive>` and `<reply>` activities) and thus parts of it are and should not be mapped to the strategic models. A major disadvantage are the poor means to describe the (local) internals of a process. Their effect can only be represented via an `<assign>` activity and its duration via a `<wait>` activity. Thus we expect the modeler to detail the internal behavior manually if such knowledge is available. In particular BPEL does not support the soft choice of process paths. If there are different paths to choose from the conditions are deterministic. Something comparable to the i* goal element is missing. If such considerations do make sense for the described process the modeler has to add them manually. It might also be the case that some deterministic choices can be relaxed to such more free choices. The lack of a goal-like construct causes also the lack of softgoal-like aspects and corresponding contributions.

The other way round we find some constructs in BPEL which are most suitable but currently not supported by SNet. Especially all the sophisticated means to handle (run-time) fault behavior which we had to omit here. We are looking forward to incorporate this and especially connect these behavioral rules to our monitoring component (cf. [6]). Concerning the `<while>` construct we are currently investigating whether we shall incorporate this already on the i* level. On

the side of ConGolog a corresponding construct (with the same name) of course already exists.[8]

Eventually we want to emphasize a nice property of BPEL which is the inherent subjective view from which the processes are described. While this can also cause some problems for deriving a combined model it nicely stresses that the whole modeling procedure is subjective: the business partner who is modeling the scenario can include only as many details about its partners and other enterprises involved as he knows.

## 5 Discussion and Conclusion

There exists already an approach [14] to map BPEL descriptions to Petri nets with the intended goal to allow for a detailed analysis. Although we think that a formal and exact transformation of BPEL into ConGolog is possible and comparable to the one into Petri nets we concentrated here on the strategic and hence more abstract level of extended i*. The reason for this is that we do not want to debug BPEL descriptions but to use them for building up models to run long-term prospective studies of the evolution of inter-organizational networks.

Another branch of work concerning the area of "adapting Golog for composition of semantic web services" is carried out by Sheila McIlraith and others [15]. They have shown that Golog might be a suitable candidate to solve the planning problems occurring when services are to be combined dynamically at run-time. Additionally they related their work also to BPEL(4WS) [16] explicitly by stating that the semantic web efforts in the research area are disconnected from the seamless interaction efforts of industry and thus propose to "take a bottom-up approach to integrating Semantic Web technology into Web services". But they mainly focus on introducing a semantic discovery service and facilitate semantic translations.

To summarize we have seen that it is indeed possible to use BPEL process descriptions for building up extended i* models. BPEL provides several constructs which are similar to the ones used in i* as for example service definitions correlate with roles, `<partnerLinkType>`s with strategic dependencies, `<partner>`s with positions. Furthermore the structuring of the process body via `<sequence>`, `<flow>` etc. can be mapped to ordering constraints of sub tasks in ext. i*. The modeler might have to provide additional information to nicely group tasks to complex sub tasks and neither goals nor softgoals with contributions result from a BPEL description. Such more strategic considerations must be added to the very detailed, executable process description. Other aspects like fault behavior and the `<while>` construct are not yet covered in SNet but would fit in.

For the future we expect to be able to provide a better support for deriving i* models from BPEL descriptions by generating corresponding Telos frames (textual representation of i* models) automatically. And of course we are looking forward to apply the transformation procedure to real world BPEL descriptions.

[8] The `<switch>` construct can be mapped to a task decomposition of 'if needed' sub tasks with suitable precondition/effect elements checking the conditions.

Additionally, we have to watch out also for the rivals of BPEL as there are BPML, BPMN, WS-CDL, etc. maybe also enabling to derive useful information from descriptions written in these languages.

*Acknowledgment.* This work was supported in part by the Deutsche Forschungsgemeinschaft in its Priority Program on Socionics, and its Graduate School 643 "Software for Mobile Communication Systems".

## References

1. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.: Continuous requirements management for organization networks: A (dis)trust-based approach. Requirements Engineering Journal, Special Issue RE'01, Springer **8** (2003) 4–22
2. Yu, E.: Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto (1995)
3. de Giacomo, G., Lespérance, Y., Levesque, H.: ConGolog, a concurrent programming language based on the situation calculus: language and implementation. Artificial Intelligence **121** (2000) 109–169
4. Jarke, M., Eherer, S., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M.: ConceptBase - a deductive object base for meta data management. Journal of Intelligent Information Systems, Special Issue **4** (1995) 167–192
5. Gans, G., Jarke, M., Lakemeyer, G., Schmitz, D.: Deliberation in a modeling and simulation environment for inter-organizational networks. In: Proc. of CAiSE03. LNCS 2681, Klagenfurt, Austria (2003) 242–257
6. Gans, G., Schmitz, D., Jarke, M., Lakemeyer, G.: SNet reloaded: Roles, monitoring and agent evolution. In: Proc. of AOIS@AAMAS-2004 Workshop. (to appear)
7. Andrews, T., et al.: Business process execution language for web services, IBM, version 1.1, 2nd public draft release. www.ibm.com/ developerworks/ webservices/ library/ ws-bpel (2003)
8. Arkin, A.: Business process modeling language 1.0. Technical report, BPMI Consortium, http://www.bpmi.org/ (2002)
9. Kavantzas, N., Burdett, D., Ritzinger, G.: Web services choreography description language 1.0. working draft. http://www.w3.org/TR/ws-cdl-10/ (2004)
10. Petrie, C., Bussler, C.: Service agents and virtual enterprises: A survey. IEEE Internet Computing (2003) 68–78
11. Liu, L., Yu, E.: Organziation Modeling Environment (OME). WWW ([Accessed 2004/08/18]) http://www.cs.toronto.edu/km/ome.
12. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos - representing knowledge about information systems. ACM Transactions on Information Systems **8** (1990) 325–362
13. McCarthy, J.: Situations, actions and causal laws. Tech. report, Stanford (1963) Reprinted 1968 in Minsky, M.(ed.): Semantic Information Processing, MIT Press.
14. Vidal, J.M., Buhler, P., Stahl, C.: Multiagent systems with workflows. IEEE Internet Computing (2004) 76–82
15. McIlraith, S., Son, T.C.: Adapting Golog for composition of semantic web services. In: Proc. of the 8th Int. Conf. on Knowledge Representation and Reasoning (KR2002). (2002) 482–493
16. Mandell, D.J., McIlraith, S.A.: Adapting BPEL4WS for the semantic web: The bottom-up approach to web service interoperation. In: Proc. of the 2nd Int. Semantic Web Conf. (ISWC). (2003)