36 A PROCESS MODEL FOR COLLABORATIVE PROBLEM SOLVING IN VIRTUAL COMMUNITIES OF PRACTICE

María Clara Casalini, Tomasz Janowski, Elsa Estevez[1]

The United Nations University
International Software for Software Technology
(UNU-IIST), Macau
{mcc,tj,elsa}@iist.unu.edu

We present a model for collaborative problem solving in Virtual Communities of Practice. The model applies a repository, comprising resources, properties and statements, to underpin the process of problem-solving. The process allows for formulating, exploring, matching and gradually refining abstract problem descriptions (one kind of resource) into the corresponding concrete solutions (another kind of resource), expanding the underlying repository with new resources, properties and statements in the process. The model is defined formally, its usefulness is argued with a simple case study and a possible implementation is described using Semantic Web.

1. INTRODUCTION

Virtual Communities of Practice (VCPs) enable distributed knowledge workers to share experience and seek solutions to concrete problems in a given field of interest, all through computer-supported interaction and collaborative work. VCPs allow members to develop individual performance and establish best practices in the field.

The extent of computer support for VCPs differs depending on the nature, scope and field of interest, ranging from member registration, to collaborative review and version control. Both synchronous interactivity (chat rooms) and asynchronous (email forums) are supported. However, while technical functions are generally supported, supporting creative aspects of community work is difficult. One reason is the shortage of methods to carry out creative activities through computer-supported processes. This is not surprising - creativity is inherently hard to formalize!

This paper presents a process-oriented model for collaborative problem solving in Virtual Communities of Practice. The model describes a systematic process of solution-building for a given problem description. The model relies on a repository to keep the record of various kinds of web resources - publications, projects, case studies, problems, solutions, etc.; properties – data about or relations between resources; and statements. Statements are triples of a subject (resource), property and object (resource or data). Problem-solving is carried out in six stages:

[1] On leave of absence from Universidad Nacional del Sur, Bahía Blanca, Argentina.

1) *problem description* – A member formulates a problem description from its own practice and adds this description to the repository as a resource.
2) *problem exploration* – Members explore the problem and gradually add to the repository the relevant resources, properties and statements.
3) *problem matching* – Members try to match the problem against similar problems described in the repository, including solved and unsolved problems.
4) *solution design* – Members decompose the problem into a number of sub-problems and add each to the repository as a problem description.
5) *solution refinement* – Members gradually add new resources, properties or statements relevant to the solution obtained so far, and link the solutions to sub-problems with the solution design, when available.
6) *solution deployment* – When all sub-problems are solved, the member adds a solution statement relating the abstract problem with the concrete solution.

The model is described conceptually then formalized using RSL (George, 1992). The repository and operations invoked by members as part of the problem-solving process are formalized. Thereafter, we discuss a possible implementation relying on Semantic Web. The usefulness of the model is argued through a case study – design an XML language and software to write and display presentation slides.

The rest of the paper is organized as follows. Section 2 provides background information in three areas related to the paper. Section 3 presents a process model including: concepts (Section 3.1), formalization (Section 3.2) and implementation (Section 3.3). Section 4 presents a case study and Section 5 draws some conclusions.

2. BACKGROUND

This section provides a brief account of the three areas related to this paper: Virtual Communities of Practice, Collaborative Problem Solving and Semantic Web.

Virtual Communities of Practice: A Virtual Community is an aggregation of individuals or businesses interacting around a shared interest, where interactions are supported by technology and guided by some norms (Porter, 2004). A Community of Practice (CoP) is a group of people sharing a concern for something they do and learning how to do it better by interacting regularly (Wenger, 2004). A Virtual Community of Practice is a CoP where interactions are supported by technology.

Following (Porter, 2004), five attributes characterize VCPs: purpose - the interest shared by members; place - the virtual space comprising the members' sense of presence; platform – the nature of interactions between members, whether synchronous, asynchronous or both; population - pattern of interaction according to the structure and social ties of the group; and profit - the economic value that a community may produce. Following (Cambridge, 2005), the lifecycle of a community comprises: (1) inquiry - identifying the audience, goals and vision of the community, (2) design – defining activities and roles to support community's goals, (3) prototype – piloting the community with selected stakeholders, (4) launch – rolling out the community to a broader audience, (5) growth - encouraging members and newcomers to participate and get engaged in the activities, and (6) sustain – cultivating and assessing the knowledge created by the community.

Collaborative Problem Solving: We defined a Problem as an intricate unsettled question. Problem-solving deals with the processes involved in finding solutions to problems. Collaborative problem-solving is problem-solving done by peers, performing the same actions, having a common goal and working together (Dillenbourg, 1999). Collaboration is the process of intertwined activities of two or more actors while solving a problem. Direct collaboration is realized by communications between actors. During indirect collaboration the actors apply the knowledge other actors made available in a repository (Conen, 1996).

Semantic Web: The Web is essentially a repository of documents aimed at human consumption. Semantic Web extends the Web by capturing formally the semantics of documents, thus automating the discovery, integration and reuse of documents (Berners-Lee, 2001). Semantic Web is implemented using several technologies and standards. For instance, Resource Description Framework (RDF) is used to specify resources (W3C, 2005). Another technology is the Web Ontology Language (OWL), used to describe relationships between resources (W3C, 2004).

RDF is a language for representing and exchanging metadata about the resources on the Web, with precise syntax and semantics. Syntactically, RDF applies an XML language called RDF/XML. Semantically, it describes web resources in terms of properties and their values. Resource descriptions are called statements and consist of a subject, predicate and object. The subject identifies the resource, the predicate determines a property of the resource, and the object defines the value of the property. Resources and properties are identified through URIs.

An ontology is a specification of a set of concepts and their relationships in a particular domain of knowledge (Gruber, 1993). Ontologies are usually expressed in a particular logic language enabling the definition of classes, properties and their relationships. For instance, OWL can be used to define ontologies. OWL is the language developed by W3C as a standard for the Semantic Web, with several tools existing to define, encode and develop ontologies.

3. COLLABORATIVE PROBLEM SOLVING FOR VCPs

This section presents the main contribution of this paper – the process for Collaborative Problem-Solving in Virtual Communities of Practice.

3.1 Conceptual Model

The process of Collaborative Problem Solving relies on a repository, with an underlying ontology, owned and developed by the community. The components in the repository are: *resources* – community assets categorized into classes, *properties* – relations between resources or data about resources; and *statements* – expressions about resources and their properties. Properties are binary relations between pairs of resources or between resources and simple values. Statements are triples of a subject (resource), property and object (resource or data), written `[sub,prop,obj]`. Figure 1 depicts the resources stored in the repository and how they are used to build solutions. Solutions can be composed by any type of resource, for instance a solution to a previous problem can be used to build a solution to a new problem.

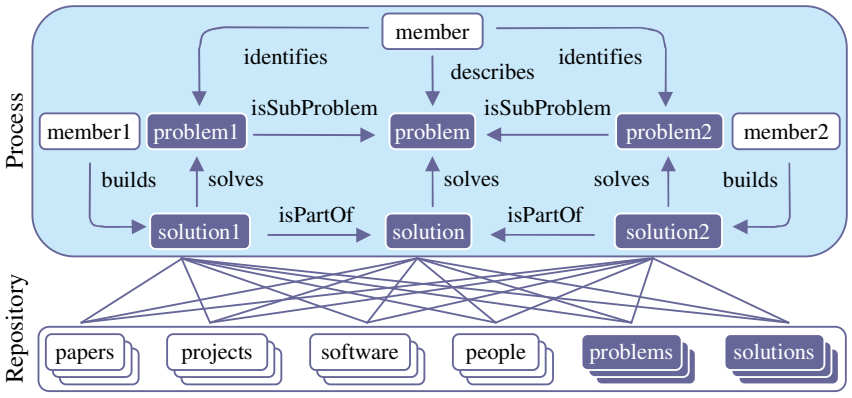

Figure 1: Repository for Collaborative Problem Solving

The process is carried out in six phases: *problem description* – adding a problem resource to the repository; *problem exploration* – analyzing the problem by adding statements linking it to existing or new resources; *problem matching* – identifying similarities between the problem and other problems in the repository by adding relevant statements; *solution design* – decomposing the problem into sub-problems and adding them as new resources to the repository; *solution refinement* – adding new statements, resources and properties related to the solution, and linking solutions to sub-problems with the solution design when they become available; and *solution deployment* – adding a solution statement when all sub-problems are solved.

Figure 2 depicts the process and how the number of statements about the problem increases during all phases. It also shows how the number of unsolved sub-problems increases during *solution design* and decreases during *solution refinement*. The process can be carried out fully collaboratively since sub-problems can be assigned to different members, who in turn apply the same process for solving them.

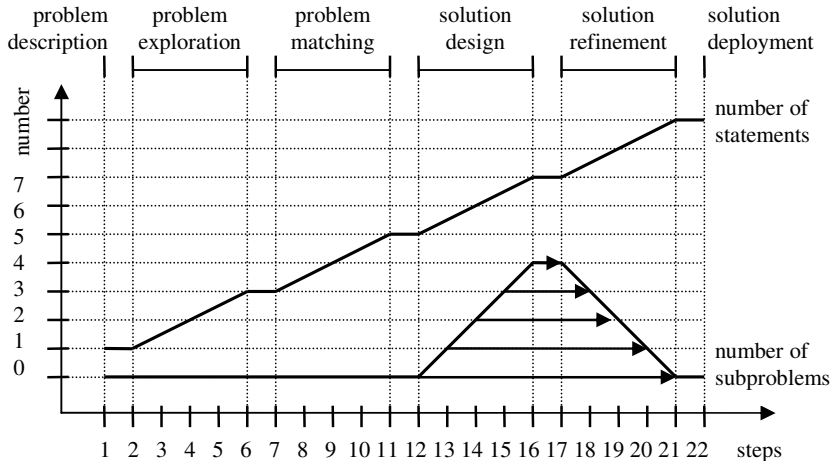

Figure 2: Process for Collaborative Problem Solving

3.2 Formal Model

The aim of this section is to show how it is possible to formalize the model described in Section 3.1. First, we introduce abstract types to represent data, resources, properties and statements. `Value` type is defined to comprise plain data or complex data (resource). In addition, three functions are defined to return a subject (`sub`), predicate (`pred`) and object (`obj`) parts of a statement.

```
type
  Data,
  Resource,
  Property,
  Statement
```

```
type
  Value == plain(Data) | complex(Resource)
value
  sub: Statement -> Resource,
  pred: Statement -> Property,
  obj: Statement -> Value
```

A repository is modeled using a type `Repos`, with functions defined to return sets of resources, properties and statements. The type is constrained to ensure that all statements in the repository only use the existing resources and properties.

```
type
  Repos',
  Repos = {| r: Repos' :- iswf(r) |}
value
  res:  Repos' -> Resource-set,
  prop: Repos' -> Property-set,
  stat: Repos' -> Statement-set
```

```
value
  iswf: Repos' -> Bool
  iswf(r) is all s: Statement :-
    s isin stat(r) =>
    sub(s) isin res(r) /\
    pred(s) isin prop(r) /\
    case obj(s) of
      plain(_) -> true,
      complex(o) -> o isin res(r)
    end
```

A number of functions are defined to modify the repository. Among them are the functions to add resources, properties and statements. All functions must ensure that the resulting repository is well-formed according to the constraint above.

```
value
  addRes: Resource >< Repos -~-> Repos
  addRes(o, r) as r' post
    res(r') = res(r) union {o} …
    pre o ~isin res(r),
  addStat: Statement >< Repos -~-> Repos
  addStat(s, r) as r' post
    stat(r') = stat(r) union {s} …
    pre canAddStat(s, r),
  addProp: Property >< Repos -~-> Repos
```

```
value
  canAddStat: Statement >< Repos -> Bool
  canAddStat(s, r) is
    s ~isin stat(r) /\
    sub(s) isin res(r) /\
    pred(s) isin prop(r) /\
    case obj(s) of
      plain(_) -> true,
      complex(o) -> o isin res(r)
    end
```

A problem is a particular kind of resource. We introduce the types representing resource and property types, along with functions `hasType` from values to types. `Problem` type is defined as a subtype of `Resource`. A function is also introduced to return a set of properties used in the statements about a given problem.

```
type
  ResType,
  PropType
value
  problem: ResType,
  exactProp: PropType
value
  hasType:
    Resource -> ResType
  hasType:
    Property -> PropType
```

```
type
  Problem = {| o: Resource :- hasType(o) = problem |}
value
  isProblem: Problem >< Repos -> Bool
  isProblem(p, r) is
    p isin res(r),
  problemProp: Problem >< Repos -~-> Property-set
  problemProp(p, r) is
    { t | t: Property :- exists s: Statement :-
      s isin stat(r) /\ p = sub(s) /\ t = pred(s) }
    pre isProblem(p, r)
```

In the following, we introduce example functions supporting various phases of the problem-solving process. In problem description, we only add the problem resource.

```
value
  describe: Problem >< Repos -~-> Repos
  describe(p, r) is addResource(p, r) pre ~isProblem(p, r)
```

During problem exploration, new statements about the problem are repeatedly added to the repository, along with new properties and resources as required. For instance, function `explore` is defined to add a single statement.

```
value
  explore: Problem >< Statement >< Repos -~-> Repos
  explore(p, s, r) is
    let r' = if pred(s) isin prop(r) then r else addProp(pred(s), r) end in
      case obj(s) of
        plain(_) -> addStat(s, r'),
        complex(o) ->
          if o isin res(r') then addStat(s,r') else addStat(s, addRes(o,r')) end
      end
    end pre sub(s) = p /\ isProblem(p, r)
```

During problem matching, the problem is compared to existing problems and suitable statements are added to record the similarities. One measure of similarity is that the two problems use the same set of properties in the statements about them. A property type `exactProp` has been defined to express this. The following function `matchExact` adds the corresponding statement, if applicable.

```
value
  matchExact: Problem >< Problem >< Repos -~-> Repos
  matchExact(p1, p2, r) is
     let s: Statement :-
       sub(s) = p1 /\ pred(s) = exactProp /\ obj(s) = complex(p2)
     in addStat(s, r) end
     pre isProblem(p1, r) /\ isProblem(p2, r) /\
         problemProp(p1, r) = problemProp(p2, r)
```

The remaining phases of problem-solving include solution design, refinement and deployment. Specific functions are defined to support these phases. For solution design, the functions add new problems as sub-problems of the original problem. For solution refinement, the functions add new statements, properties and resources related to the solution, as well as functions relating the solution to the solutions of sub-problems. For solution deployment, the function adds a statement relating the problem and the final solution, when all sub-problems are eventually solved.

3.3 Implementation

The process described above is currently implemented to support the community portal for the UNeGov.net initiative – Building a Community of Practice for Electronic Governance (UNeGov.net, 2006). The portal is developed in Java using the Jena Semantic Web Framework (Jena, 2006). The portal maintains the database of members and the repository relies on the ontology described in OWL. Jena API is used to manipulate the ontology, browse and search the repository, and maintain the knowledge created by the community. All resources are represented by URIs. For papers or software resources, the URIs link to their electronic versions. For member or organization resources the URIs link to their home pages.

4. CASE STUDY

Consider a Virtual Community of Practice focused on XML technology and software development. Suppose this VCP is presented with the following problem P: "Define an XML language and software to write and display presentation slides".

Suppose the repository contains the resources of the types `language`, `tool`, `element` and `schema`, problem `P'` with some statements, and its solution `S'`:

`P'`: Define an XML Language for specifying contents of documents.
`S'`: XML4Doc specifies contents of documents.
`[P',about,schemas], [P',includes,title], …`

The problem-solving process is depicted in Figure 3. The process begins by adding `P` as a `problem` resource to the repository (phase 1). During problem exploration (phase 2) we add statements `[P,about,schemas]` and `[P,about,tools]`. Similarities identified throughout problem matching (phase 3) are expressed by `[P,subProp,P']` - P contains some but not all properties of `P'`. Three sub-problems are identified during solution design (phase 4):

`P1`: Analyze existing presentation slides to identify typical elements.
`P2`: Define an XML Schema to describe the contents of slides.
`P3`: Select a tool for the presentation of slides.

Once the sub-problems are identified, similar problem-solving is carried out for each of them (see Figure 3). `P1` is solved with `S1` identifying presentation elements found in the analyzed slides. Several statements are added to relate `P2` to these elements. As the same elements were found in documents, `P2` is matching the properties and values of `P'`, therefore its solution `S'` is also suitable for `P2`. `P3` is further sub-divided into `P4` – transform XML to HTML, and `P5` – select a tool to display HTML. `P4` is solved by the XSLT transformation for the schema obtained in `S'`, while `P5` is solved through browsers recorded as tools in the repository.

Phase	Problem	Sub-Problems		
1	P:define slides	P1:analyze slides	P2:define schema	P3:select tool
2	[P,about,schemas] [P,about,tools]	[P1,about,elements]	[P2,about,schemas] [P2,includes,title]	[P3,about,tools]
3	[P,subProp,P']	[P1,subProp,P'] [P1,exactProp,P]	[P2,exactProp,P'] [P2,exactValue,P']	[P3,exactProp,P] [P3,subProp,P2]
4	P1:analyze slides P2:define schema P3:select tool			P4:XML to HTML P5:HTML tools
5	S:XML4Doc/browser [Si,part,S] i=1..3	S1:slides analyzed		S3: HTML browser [Si,part,S3] i=4,5
6	[S,solves,P]	[S1,solves,P1]	[S',solves,P2]	[S3,solves,P3]

Figure 3: XML Case Study – Problem Solving and Sub-Problem Solving

During solution refinement (phase 5) for `P`, `S`: "XML4Doc is a schema for slides and any browser can display them." is added. Statements `[Si,part,S]` for `i = 1..3` are added to relate sub-problem solutions with S. Finally, during solution deployment (phase 6), `[S,solves,P]` is added to link P with its solution S.

5. CONCLUSIONS

The main objective of this paper was to present a process-oriented model for cooperative problem solving in Virtual Communities of Practice, suitable for implementation as part of community portals. This was motivated by the shortage of computer-supported methods to support creative aspects of community work.

The process relies on a repository containing resources, properties and statements. Problems, solutions and partial solutions are all expressed as resources. Problem-solving is about connecting a problem description to the existing resources, creating new statements, properties and resources in the process. In particular, new sub-problems are identified and solved through the same six-phase processes. The paper formalized the model, indicated a possible implementation, and illustrated its usefulness through a case study in XML language/tool development.

The main benefit of the approach is the definition of a systematic process to carry out problem-solving in any domain of knowledge, enabling computer support to organize the process and maintain the repository. At the same time, the approach requires that problem-solving conforms to a particular formal structure, and the suitability of this structure for particular problem domains is yet to be investigated.

Future work includes expanding the model to cover community concepts, exploring opportunities for automation in different phases, implementing the model and assessing its effectiveness in various domains of knowledge.

ACKNOWLEDGEMENTS

We would like to thank Irshad Khan, Adegboyega Ojo and Gabriel Oteniya for useful discussions and comments about this work.

REFERENCES

1. Berners-Lee Tim, Hendler James, Lassila Ora. The Semantic Web. Scientific American, May 2001.
2. Cambridge Darren, Kaplan Soren and Suter Vicki. Community of Practice Design Guide, 2005. http://www.educause.edu/ir/library/pdf/NLI0531.pdf.
3. Conen Wolfram, Neumann Gustaf. Prerequisites for Collaborative Problem Solving. Proceedings of WETICE 96, IEEE 5th Intl. Workshops on Enabling Technologies, Stanford, CA, June 1996.
4. Dillenbourg Pierre. What Do You Mean By "Collaborative Learning". Collaborative-Learning: Cognitive and Computational Approaches, pp 1-19. Oxford: Elsevier, 1999.
5. George Chris, et al. The RAISE Specification Language. Prentice Hall, 1992.
6. Gruber Thomas. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer, August 1993.
7. Jena. Jena - A Semantic Web Framework for Java, 2006. http://jena.sourceforge.net/.
8. Porter Constance Elise. A Typology of Virtual Communities: A Multi-Disciplinary Foundation for Future Research. Journal of Computer-Mediated Communication. Vol. 10 (1), 3, November 2004.
9. UNeGov.net. Community of Practice for Electronic Governance, 2006. http://www.unegov.net.
10. Wenger Etienne. Communities of Practice - A Brief Introduction, June 2004. http://www.ewenger.com/theory/index.htm.
11. W3C. Technology and Society Domain, Semantic Web Activity. Resource Description Framework, October 2005. http://www.w3.org/RDF/.
12. W3C. OWL Web Ontology Language Overview. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-features/.