Experiences Developing a Requirements Language Based on the Psychological Framework *Activity Theory*

Geri Georg[1] and Lucy Troup[2]

[1]Computer Science Department, Colorado State University, Fort Collins, USA
[2]Psychology Department, Colorado State University, Fort Collins, USA
georg@cs.colostate.edu
Lucy.Troup@ColoState.edu

Abstract. We have developed a Domain Specific Language (DSL) for requirements elicitation that is based on the psychological framework of Activity Theory (AT). AT emphasizes the social context in which human activity takes place, and thus is useful to systematically develop models of social contexts, validate these contexts with stakeholders, and identify potential sources of system evolution based on identified changing social constraints. AT holds potential as a requirements elicitation tool for complex human interactive systems with a diverse set of stakeholders that do not have common goals. Our adaptation of AT for use in software engineering has evolved over time as we have used it in a case study and developed limited tools that can support designers both during initial system design and during system evolution. Here we describe how the USE tool was applied to develop the DSL and how we have used this tool to create instances of AT models and analyze them for structural constraint inconsistencies. We identify some of the issues encountered in this process and some of the remaining open issues regarding a USE model as an implementation of our DSL.

Keywords: Activity Theory; DSL; USE tool; Modeling Social Behavior

1 Introduction

An understanding of a system's *social* context is needed to identify the socio-related functionality, security, performance, and other system concerns that impact the usability and acceptability of complex systems that comprise both human and computing elements. This social context describes the evolving bi-directional impact and relations between a system and the community in which it is developed and deployed. A more complete system design that addresses the social constraints of the system can be critical to its ultimate success. However, the problem of systematically modeling the social context and being able to use this information in requirements elicitation and early design remains an area of research.

We have approached the problem of social constraint specification and validation through the application of the Activity Theory (AT) psychological paradigm [2]. AT provides concepts and terminologies that are easy to understand by a wide range of stakeholders within a framework that encourages designers to explore a system's social context. The framework can help requirements engineers formulate important questions to elicit both explicit and implicit constraints in the social environment.

However, AT is not formalized and, while understandable to the general population, its efficacy relies heavily on the skills of the analysts using it. We have therefore defined an AT domain specific language (DSL) [18], designed to add rigor to the basic elements of the framework and their inter-relationships. An interesting issue we have identified is that while the framework encourages thinking about a system in a holistic sense, it can also lead to inconsistencies in requirements documentation (as models) or understanding since in fact it is so flexible and dependent on human input. We have therefore added further constraints to our language to help ameliorate some of these inconsistencies.

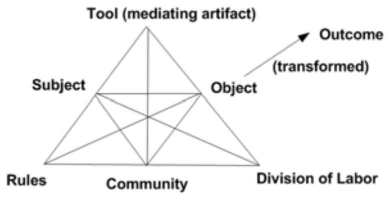
The contribution of this paper is to demonstrate how we have applied the USE tool [17] during the creation and evolution of the AT DSL, and how we have used it to create AT model instances that can be analyzed for structural and constraint inconsistencies often found in AT models. This work reports on a feasibility experiment regarding the USE tool as a mechanism to realize our AT language. While we point out issues we have encountered during this process, we do not discuss alternative methods or tools that could be used to address them.

In Section 2, we describe AT and some issues with using it in software development. Section 3 describes our initial and evolutionary efforts at defining an AT DSL, and how the USE tool fit into this process. Section 4 demonstrates the USE tool analysis of an example AT instance model and describes issues related to the USE tool that we consider outstanding research issues. Related work is described in Section 5, and our conclusions and future work are presented in Section 6.

2 Activity Theory

AT was initially proposed [11][19] to aid exploration of the complex social relationships inherent in any human activity. It was extended [2] and has also become an important theoretical framework in the field of Human Computer Interaction (HCI) [6]. AT is useful in situations where it is necessary to explore the diverse and complex social context of a system. This is especially critical in socio-technical systems with both human and computing components since their success is often contingent on how well they demonstrate a thorough understanding of, and support of, social constraints.

AT defines human activity as systems of several elements and their mediating relations. An activity system evolves over time to relieve the tensions caused by contradictions in the system. Engeström [2] identified different types of contradictions, including those within AT elements and those between different AT elements of a single activity system.



The figure to the left shows the diagrammatic form of an activity system developed by Engeström. We term this diagram an Activity System Diagram, or ASD. The *object* of a human activity is transformed into the *outcome* by the *subject*, using the mediating *tool*. For readers with software development backgrounds, the term ***object*** may be confusing. However, these terms have been used in the AT literature since the original writings, so we will use the term *object* in the AT sense in this paper. When needed, we will use the term ***software object*** to indicate the term in the software development sense.

Both the *object* and *tool* can be physical or conceptual, and there can be multiple of these and the *subject* elements. The *community* is anyone sharing the same *object*. Relations between the *subject* and *community* members are mediated by *rules*, which can be implicit or explicit norms and conventions. The *division of labor* (DoL) mediates relations between the *object* and *community* members. *DoL* items specify how the task of transforming the *object* into the *outcome* is distributed across the *community*. AT recognizes that human activity rarely takes place in isolation, so it includes the concept of *activity system networks*, where multiple ASDs are connected. Networks occur when the outcome of one ASD is used as an element in another ASD (e.g. as a tool). We term this type of relation a *network relation* in our AT language.

The AT framework is very flexible, and is applicable in complex social situations to explore implicit as well as explicit relations and interactions among the participants of an activity [7][14]. In addition, the concepts and terminology are readily understood by a wide variety of stakeholders with diverse backgrounds. AT is therefore well-suited to systematic application in Requirements Engineering (RE), and can be useful to identify the social constraints that must be taken into account in the design of complex socio-technical systems. In particular, discussions that identify mediating *rules* and *DoL* items can elicit the often implicit goals and actions of stakeholders.

Issues using AT in software development. There are several issues that arise when AT is used in RE. First, when there are multiple rules and community members, it is not possible, from Engeström's ASD, to identify a particular rule that mediates between a particular community member and the subject. In general, any time there are multiple DoL items, Tools, or Rules, if there are multiple subjects, objects, or community members, then it is not possible to determine the relevant mediat*ing* element item (i.e. Tool, Rule, or DoL) for a particular pair of mediat*ed* element items (i.e. Subject, Object, or Community member). Our language addresses this issue through explicit mediation relations described in the next section.

A second issue when using AT in software development is there may be ASD elements identified by the stakeholders that are not directly related to the current activity object. These inconsistencies can lead to over- or under-specified ASDs and can only be avoided through experience. We address this problem with constraints on language concepts. A third issue related to ambiguity is that the ASD network relation is very informal in the AT framework in the sense that while the relation must involve the outcome of one ASD, it is not specified what the target ASD element should be. Our language constrains these target elements. A fourth issue of incompleteness is that multiple ASDs can also represent evolution of an activity over time, thus implying a temporal relationship but the framework does not specify how this relationship is to be described. We have not yet addresses this temporal relationship in our AT language.

Finally, to best use AT as a requirements elicitation tool, it is necessary to be able to use its results in system design. Our work includes formalized trace-links between the AT language classes and relationships and relevant portions of the goal and scenario design modeling language standard User Requirements Notation (URN) [8] that allow us to partially automate bi-directional transformations between ASD network models and URN goal models in the jUCMNav tool[10]. The jUCMNav tool maintains traces between goal models and use case map scenarios that serve as high level design models. We are thus able to move between AT and design models through these traces. Details of this 'requirements to design' process are beyond the scope of this paper. However, we have applied these design links to a case study associated with a data capture and interpretation system designed for use in vector-borne disease control in Mexico. A technical report is available with these detailed examples [5], however in this paper we provide a very simple example to demonstrate the AT DSL, and do not discuss the transformation to design further.

3 AT Language Development

Initial version of the AT DSL. The initial version of the AT DSL was developed using a metamodel that defined the ASD elements and some simple Object Constraint Language (OCL) constraints [15][16]. For example, an *ASD* class was related to an abstract class *Element* which was specialized into all seven AT concepts. The relation multiplicity was defined as 7..* on the *Element* role, and 1..* on the *ASD* role. An additional well-formed OCL constraint was added that there must be at least one of each type of specialization in the *Element* role set.

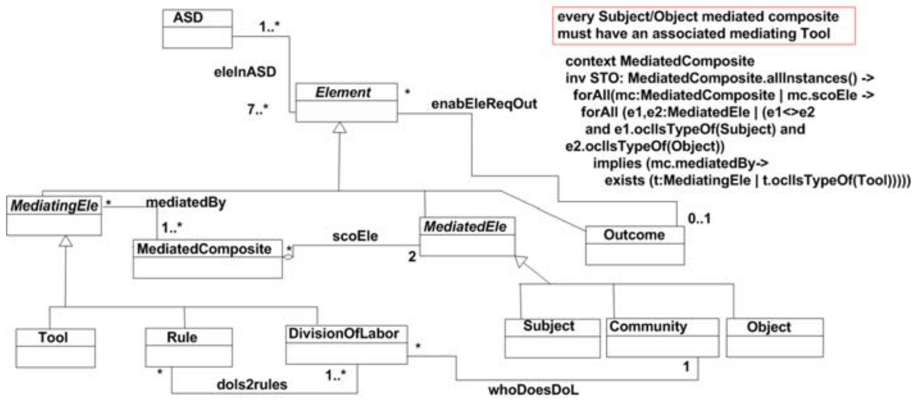
A network relation between an outcome of one ASD and an element of another is part of the AT framework, and this relation was further constrained in that the element of the second ASD must not be an object or outcome of the second activity. This constraint is possible because from experience, it makes no sense for an object or outcome of an activity to be the outcome of a previous activity. The *network* relation and its constraint decrease the ambiguity of the network concept in the AT framework. The initial language also contained the explicit notion of a hierarchical decomposition of activities in order to help structure AT models. The decomposition relation was defined between a DoL item in one ASD and another ASD that further describes this DoL item as an entire activity.

It was very easy to convert this metamodel and its constraints into a USE tool [17] model and check it for structure and constraint issues. After fixing the problems found by the tool, the result was a simple implementation of our language. We used this approach in our subsequent versions, and found that the USE implementation was easy to modify for exploration purposes regarding mediating relations and augmentation to minimize over- and under-specification.

Subsequent versions. This simple AT metamodel was augmented with the additional relations needed to reduce over-specification caused by ASD elements that are really related to a different activity and thus belong in a different ASD. This was accomplished by adding the *dols2rules* relation that makes sure every rule in an ASD is associated with some DoL item, and the *whoDoesDoL* relation, which ensures that there is some community member associated with each DoL item. If a rule exists which is not related to some DoL item, then the rule probably does not belong in the respective ASD or else a DoL item is missing. Respectively, if there is a DoL item that is not associated with some community member, there is probably a missing community member. The USE tool makes these problems explicit through its analysis capabilities, and the modeler can then decide how to address them.

Another augmentation of the language metamodel was to add explicit mediation relationships according to the AT framework. Multiple approaches were attempted, including creating ternary mediating relationships between, e.g. Subject, Tool, and Object classes. However ternary relationships made both the metamodel diagram and USE model file more complex and harder to interpret. After exploring several alternatives, three new classes were finally introduced: two abstract classes specialized from the *Element* class, for mediat*ing* (Tool, Rule, and DoL) and mediat*ed* (Subject, Object, and Community) elements, and a "composite" class to express the pair of mediated elements. The mediating elements were moved to become specializations of the *MediatingEle* class, and mediated elements moved to become specializations of the *MediatedEle* class. While this simplified the metamodel, it also meant that a relation and constraint had to be added to ensure that all the elements involved in mediation are elements of the same ASD, and that the composite (*MediatedComposite*) consists of mediated elements of different types (e.g. Subject and Object, and not Subject and Subject). This structure made it easier to consider the characteristics of AT mediation in detail. For example, since every activity consists of a subject using a mediating tool to accomplish an object, the constraint was be added that every subject/object pair needs to have a mediating tool. Over-specification is indicated by e.g., a tool that is not part of such a relationship, and under-specification is indicated by e.g., a subject/object pair that does not have an associated mediating tool.

Resulting version. The most complete DSL metamodel for AT [4], resulting from these evolutions, has 14 classes, 3 of which are abstract, 14 associations, 3 generalizations, and 9 OCL constraints. The portion of the AT language metamodel that describes the Engeström model, along with some of the relevant relations and one OCL constraint is shown in Fig. 1.



```
context MediatedComposite
inv STO: MediatedComposite.allInstances() ->
  forAll(mc:MediatedComposite | mc.scoEle ->
   forAll (e1,e2:MediatedEle | (e1<>e2
    and e1.oclIsTypeOf(Subject) and
e2.oclIsTypeOf(Object))
     implies (mc.mediatedBy->
      exists (t:MediatingEle | t.oclIsTypeOf(Tool)))))
```

Fig. 1. Portion of AT metamodel with OCL constraints

This portion of the AT language metamodel specifies framework concepts as specializations of the abstract classes *Element* and its specialized abstract classes *MediatingEle* and *MediatedEle*. It also and shows some of the relations between these concepts. For example, elements are related to ASDs (via the *eleInASD* relation). Network relations are specified via a relation between the outcome of one ASD and some other element of another ASD (*enabEleReqOut* relation with an OCL constraint that requires the OCL *type* of the enabled element to be one of the concrete types: Tool, Rule, DoL, Subject, or Community, and the element must be part of a different ASD).

The mediation-related OCL constraint shown in Fig. 1 is that the two every mediated composite consisting of a Subject and an Object must have a Tool as its mediating element. This constraint is shown next to the metamodel in Fig. 1, with an English description above it.

The complete USE model, including OCL constraints for this version of the language is 141 lines of text. Software object model ASDs have also been created adhibiting the USE tool. These software object models can be checked against the structural and OCL constraints of the language metamodel, as demonstrated below.

4 USE Analysis of ASD Instances

Due to space constraints, only a simple example of the AT DSL and its USE tool analysis are presented in this paper. This example demonstrates the activity of taking a vacation, and an ASD model is shown in Fig. 2. The figure shows both the initial version of the *Vacation ASD* and the modifications made to it as a result of USE tool automated analyses. For this example, the modifications are deletions, which are shown in Fig. 2 using double strikethrough texts.

In the initial version of the Vacation ASD, the *Subject*, a Travel Planner (***P***), uses multiple *Tools* to accomplish the *Object* (*Make arrangements to go on vacation*) of this activity, which is transformed by the activity into the *Outcome* (*All tickets, bookings in hands of, and being used by the Travel Group*). The tools include the knowledge and experience of friends (T1) and travel agents (T2), brochures (T3), the internet (T4), and reservation or booking systems (T5). The object is transformed into the outcome through many DoL tasks – ***P*** investigates options, makes decisions, purchases items, and makes personal arrangements. The Travel Agent (***A***) provides information and makes bookings. Other members of the Travel Group (***G***) must make personal arrangements e.g. packing or arranging for tasks to be done in their absences, and also check-in for the vacation. Finally, Activity Companies (***C***) provide information about their activities, advertise, and also provide the opportunities for various activities. There are three rules associated with the activity: ***P*** must communicate to the entire travel group, ***G***, the decisions that have been made: when the vacation will be, where it will take place, and what activities will be done (R1), ***A*** must provide feedback about any possible issues or problems with the decisions that ***P*** has made (R2), and ***P*** can delegate investigation/arrangements/decisions to anyone in ***G*** (R3). The Community is made up of all these participants: ***P***, ***G***, ***A***, ***C***, and friends (***F***).

To demonstrate USE analysis on this example, Fig. 3 shows the USE screen shot of the ASD model, including the results of constraint testing. The main window shows the software object model of the ASD network from Fig 2. *Software objects* (as opposed to AT objects) are displayed as rectangles labeled with an identifier and class (e.g., VACSub:Subject toward the upper left in the main window). Association instances are shown as links between software objects, labeled with the association name (e.g., *eleInASD* between VACSub:Subject and VAC:ASD, located to the right and slightly below the VACSub:Subject software object, approximately in the middle of the software object diagram). Composite relations are also shown (e.g., VACSub:Subject is a member of the MediatedComposite called VAC-PSO:MediatedComposite, located in the upper center of the diagram, along with VA-

COObj:Object which is located in the upper part of the software object diagram on its right side).

The result of the invariant evaluations are shown in the smaller window located in the same area as the software object view, titled ‘Class invariants’. One constraint fails and there are structural errors that are listed in the log area below the main window that shows messages from the tool and also the results of structural checks (log messages are not shown in Fig. 3).

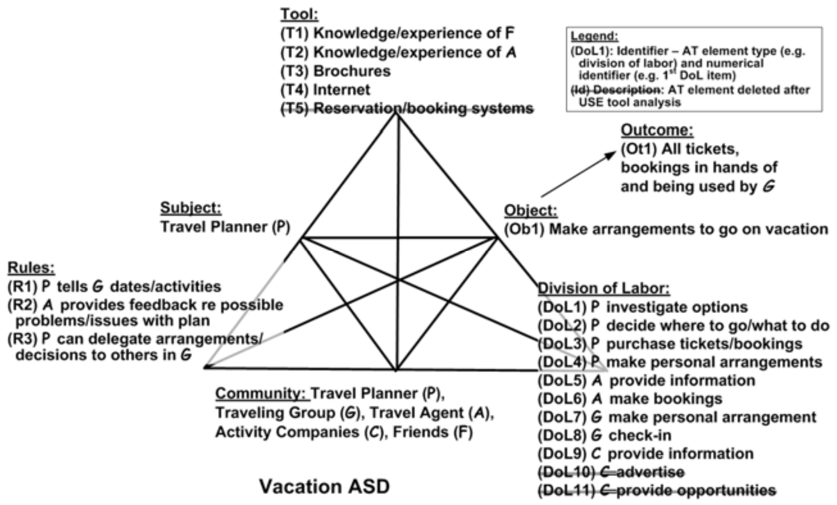

Tool:
(T1) Knowledge/experience of F
(T2) Knowledge/experience of A
(T3) Brochures
(T4) Internet
~~(T5) Reservation/booking systems~~
Legend:
(DoL1): Identifier – AT element type (e.g. division of labor) and numerical identifier (e.g. 1st DoL item)
~~(Id) Description~~: AT element deleted after USE tool analysis
Outcome:
(Ot1) All tickets, bookings in hands of and being used by G
Subject:
Travel Planner (P)
Object:
(Ob1) Make arrangements to go on vacation
Rules:
(R1) P tells G dates/activities
(R2) A provides feedback re possible problems/issues with plan
(R3) P can delegate arrangements/ decisions to others in G
Division of Labor:
(DoL1) P investigate options
(DoL2) P decide where to go/what to do
(DoL3) P purchase tickets/bookings
(DoL4) P make personal arrangements
(DoL5) A provide information
(DoL6) A make bookings
(DoL7) G make personal arrangement
(DoL8) G check-in
(DoL9) C provide information
~~(DoL10) C advertise~~
~~(DoL11) C provide opportunities~~
Community: Travel Planner (P), Traveling Group (G), Travel Agent (A), Activity Companies (C), Friends (F)
Vacation ASD

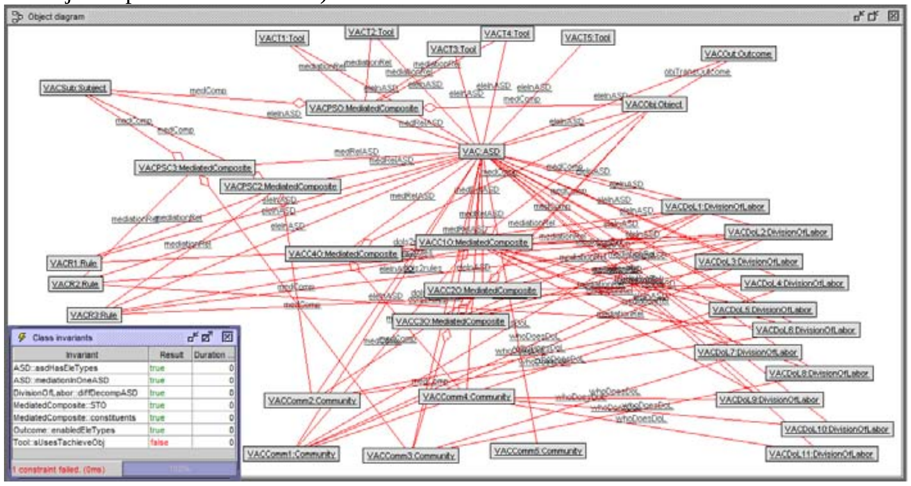
Fig. 2. Taking a vacation ASD

There are four errors indicated by the USE analysis. Three can be seen in the log error messages, and all are concerned with mediation relationships. The first logged structural error is that T5 is not involved in any mediating relationship. The other two are that DoL10 and DoL11 are not involved in any mediating relationships. These errors may indicate over-specification of the ASD, and this is how they have been interpreted. T5 is related to actually making bookings, and while the travel planner could certainly do this, the division of labor specifies that this is a responsibility of the travel agent. Thus, this tool is not used by the subject to achieve the object of the activity. It more properly belongs in an ASD that describes the travel agent activity of making bookings, that is, a hierarchical decomposition of DoL6 into its own ASD. The argument for removing the two DoL items is similar; they are really part of an activity company providing information for the travel planner, and thus probably also belong in an ASD created through hierarchical decomposition of DoL9, with the activity company as the subject.

The constraint error is related to the more general problem with T5. The specific constraint problem is that T5 is not mediating any subject-object mediated composite. The USE tool provides an evaluation browser that allows the modeler to probe for details of constraint errors which shows that there is no Subject/Object mediation specified for T5.

Open issues. While our experience with USE has been quite good to date, we are now at the point where we are exploring methods to realize AT contradiction analysis, and this entails natural language processing. One alternative is to structure natural language descriptions of ASD elements and then make OCL queries to find elements that may contain contradictions. An example would be that the Rule class might have attributes corresponding to a sentence: sentence subject, sentence object, and predicate. R1 restructured according to this convention would be "Travel planner (*P*) must communicate vacation dates and activities to the travel group (*G*)"; the sentence subject is *P*, the sentence object is *G*, and the predicate is "must communicate vacation dates and activities". A simple query might be used to find all Rules where the one of the attributes is the same, following an assumption that contradictions may occur if there is more than one Rule resulting from such a query.

Structuring these queries in OCL may not be possible for all possible contradiction conditions, and thus experimenting with methods to discover contradictions with this tool may be quite difficult. (Note that this problem would exist with any OCL tool; it is not just a problem with USE.)



Invariant	Result	Duration ...
ASD::asdHasEleTypes	true	0
ASD::mediationInOneASD	true	0
DivisionOfLabor::diffDecompASD	true	0
MediatedComposite::STO	true	0
MediatedComposite::constituents	true	0
Outcome::enabledEleTypes	true	0
Tool::sUsesTachieveObj	false	0

1 constraint failed. (0ms)

Fig. 3. USE analysis – *Vacation ASD* with checks and constraint results

Another, more general problem with the USE implementation of AT is software object model creation. Due to the mediating relationships, a command scripting file to create these models becomes quite complex. For example, the ASD shown in Fig. 2 (without the deleted items) requires a command file with 112 commands to create one each ASD, object, outcome, and subject; four tools, three rules, five community members, nine division of labor items, and the various relations. Of these, 48 commands alone are needed to create the software objects and relations for the mediating relationships. As can be seen from Fig. 3, a visualization of even this fairly simple ASD is quite complex and difficult to follow. Visualization of ASDs is critical for stakeholder validation, so that stakeholders can easily find the various AT concepts. To date our case study has uses simple drawings that are created manually, such as the one shown in Fig. 2.

One way to deal with both software object model creation and visualization complexity is to develop a user interface *wrapper* for AT to interface with the USE tool. Such a wrapper could allow an ASD diagram to be displayed as the simpler Engeström triangle such as the one shown in Fig. 2, and to use this structure to specify ASD elements. Mediated composite instances could be created by selecting ASD elements, and these could be associated with mediating element instances, again through simple selection. The wrapper would need to generate scripts to run in the USE tool to actually create the software objects in USE. This area remains an open research issue, however it severely restricts the USE model implementation of AT for general modeling purposes.

5 Related Work

Previous research has explored using activity theory in requirements elicitation [3][12][13] where AT is used to categorize requirements, and software engineering [1][9] where it is used to identify AT elements. AT has also been used in HCI to analyze human-computing interactions and identify obscured or ambiguous goals [20]. Our work differs from these approaches in that we have explicitly constrained the AT framework for systematic and repeatable use, by defining a metamodel with associated constraints and implementing the language in the USE tool where software object instances can be analyzed.

6 Conclusions and Future Work

In this paper we discuss the USE tool as we have used it to develop a requirements DSL based on the Activity Theory (AT) psychological framework. We demonstrated how the USE tool automated analysis of resulting social models can help identify inconsistencies in the models. These inconsistencies can be used in the Requirements Engineering process to identify over-specification, under-specification, and areas for additional discussion and clarification among diverse groups of stakeholders.

Our experience indicates that the USE tool is valuable for initial development and incremental extensions to our AT DSL specifications. It has also proven useful to explore structural alternatives to add AT mediation relationships to the language. USE tool suitability for our language would be greatly enhanced with an AT user interface wrapper that generates software object model scripts for the tool.

Acknowledgements: We wish to thank our collaborators, Daniel Amyot, Robert France, Saul Lozano-Fuentes, Gunter Mussbacher, and Dorina Petriu for their help in defining the AT DSL, mapping its concepts to the URN goal and design language, and using it in the case study work in Mexico. We would also like to thank the developers of the USE tool, especially Martin Gogolla, for their insights and help with modeling in the USE tool.

References

[1] P. Collins, S. Shukla, and D. Redmiles, “Activity Theory and system design: a view from the trenches”, *Computer Supported Cooperative Work*, vol. 11, no. 1-2, pp. 55–80 (2002)

[2] Y. Engeström, *Learning by expanding*, Helsinki: Orienta-Konsultit (1987)

[3] R. Fuentes-Fernández, J.J. Gómez-Sanz, and J. Pavón, “Requirements elicitation and analysis of multiagent systems using Activity Theory”, *IEEE Transactions on Systems, Man, and Cybernetics*—Part A: Systems and Humans, vol. 39, no. 2, pp. 282–298, March (2009)

[4] G. Georg and R.B. France, *An Activity Theory Language: USE Implementation*, Colorado State University Computer Science Department Techn. Report, CS-13-101, http://www.cs.colostate.edu/TechReports/Reports/2013/tr13-101.pdf (acc. Mar. 2013)

[5] G. Georg and G. Mussbacher, *SE Tool Analysis of Activity Theory Models*, Colorado State University Computer Science Department Techn. Report, CS-13-102, http://www.cs.colostate.edu/TechReports/Reports/2013/tr13-102.pdf (acc. Mar. 2013)

[6] H. Hasan, “Integrating IS and HCI using Activity Theory as a philosophical and theoretical basis”, *Australasian Journal of Information Systems* (AJIS), vol. 6, no. 2, pp. 44–55, May (1999)

[7] M. Hasu and Y. Engeström, “Measurement in Action: An activity-theoretical perspective on producer-user interaction”, *Int. Journal on Human-Computer Studies*, vol. 53, no. 1, pp. 61–69 (2000)

[8] ITU-T, *User Requirements Notation (URN) – Language definition*, ITU-T Recommendation Z.151 (10/12), Geneva, Switzerland, October 2012; http://www.itu.int/rec/T-REC-Z.151/en (acc. Mar. 2013)

[9] M. Korpela, H.A. Abimbola, and K.C. Olufokunbi, “Activity analysis as a method for information system development”, *Scandinavian Journal of Information Systems*, vol. 12, pp. 191–210 (2000)

[10] jUCMNav website, http://softwareengineering.ca/jucmnav (accessed Mar. 2013)

[11] A.N. Leont’ev, *Activity, consciousness, and personality*, Englewood Cliffs, NJ: Prentice-Hall (1978)

[12] L.E.G. Martins, “Activity Theory as a feasible model for requirements elicitation processes”, *Scientia Interdisciplinary Studies in Computer Science*, vol. 18, no. 1, pp. 33–40, January/June (2007)

[13] G.C. Neto, A.S. Gomes, and J.B. Castro, “Mapping Activity Theory diagrams into *i** organizational models”, *Journal of Computer Science & Technology* (JCS&T), vol. 5, no. 2, pp. 57–63 (2005)

[14] I. Núñez, “Activity Theory and the Utilisation of the Activity System according to the Mathematics Educational Community”, special issue of *Educate*, December, pp. 7–20 (2009)

[15] Object Management Group: *Object Constraint Language (OCL) 2.3.1* (2012); http://www.omg.org/spec/OCL/2.3.1/ (acc. Mar. 2013)

[16] Object Management Group: *Unified Modeling Language (UML) 2.4.1* (2011); http://www.omg.org/spec/UML/2.4.1 (acc. Mar. 2013)

[17] USE (The UML-based Specification Environment) website, University of Bremen; http://sourceforge.net/apps/mediawiki/useocl/index.php?title=Main_Page (acc. Mar. 2013)

[18] A. Van Deursen, P. Klint, and J. Visser. “Domain-specific languages: An annotated bibliography”, *ACM Sigplan Notices*, 35(6), pp. 26–36 (2000)

[19] L.S. Vygotsky, *Mind in society: the development of higher psychological processes*, Cambridge, MA: Harvard University Press, written 1931 (1978)

[20] J.P. Zappen and T.M Harrison, “Intention and motive in information-system design: toward a theory and method for assessing users’ needs”, *Digital Cities 3: Information Technologies for Social Capital*, LNCS vol. 3081, Springer, pp. 354–368 (2005)