Agent-assisted Distributed Requirements Elicitation And Management

Chee Fon Chang, Aneesh Krishna and Aditya K. Ghose
Decision Systems Laboratory
School of Information Technology & Computer Science
University of Wollongong
NSW 2522 Australia
email: {c03, ak86, aditya}@uow.edu.au

Abstract

Requirements engineering (RE) is concerned with the elicitation of requirements from users (stakeholders), analysis of these requirements and the generation of a coherent specification for downstream software engineering processes. Present-day RE projects typically involve large, diverse and distributed groups of stakeholders, who specify their requirements mostly using informal notation. In this paper, we describe the design and implementation of the ADREAM system which addresses these issues. We propose a novel agent-mediated architecture for the ADREAM system, a set of novel requirements elicitation strategies as well as a novel mechanism for eliciting formal descriptions of informal requirements.

Keywords: Web-based Software Engineering, Software Agents, Software Tools, Framework Techniques, Formal Methods.

1 Introduction

Given the size and complexity of present-day development projects, requirements engineering researchers and practitioners are having to address several difficult challenges. Requirements are specified by multiple sets of stakeholders. Stakeholder viewpoints often contradict each other [5, 16, 4, 2]. Requirements are typically in a state of flux and requirements evolution leads to additional inconsistencies. Functional requirements may contradict other functional requirements, but may also be inconsistent with non-functional requirements and requirements rationale [7, 9]. Inconsistency handling is therefore a key component of multi-perspective requirements engineering. Inconsistency management requires some means of detecting inconsistencies as well as some form of support for resolving such inconsistencies. Current industry-standard requirements notations are typically semi-formal (such as UML) or informal (such as plain-text English). It is difficult to devise means of providing automated (or semi-automated) support for inconsistency detection and resolution in specifications written using such notations. Being able to translate such specifications into representations in some underlying formal language would simplify these tasks (via the use of theorem-provers for these languages). However, these industry-standard notations lack formal semantics, making it almost impossible to automate the process of translation. Resolving inconsistencies often involves stakeholder negotiation. Frequent changes to the specification leads to frequent re-negotiation. Requirements elicitation remains a difficult problem.

The REAGENT project seeks to develop a comprehensive agent-mediated infrastructure to support the full range of requirements engineering processes (see [6] for a brief overview). A key design principle in this approach is that each stakeholder is represented by an agent which elicits requirements (from the corresponding stakeholder), stores them and negotiates with other stakeholder agents to resolve inconsistencies and obtain a specification that all stakeholders agree to. Stakeholder agents also monitor requirements throughout the software life-cycle, generating alerts whenever any requirement managed by it that is included in the currently accepted specification happens to be violated in some downstream artefact.

The Agent-assisted Distributed Requirements Elicitation And Management (ADREAM) system forms a part of the REAGENT framework, and implements several of its key components. It implements the distributed agent infrastructure underlying the REAGENT architecture, consisting of an extensible set of stakeholder agents and a central entity that serves both as an arbitrator and a store of the currently agreed specification, referred to as the common requirements repository. It also implements the interactive, event-driven elicitation functionality encoded in each stakeholder agent. As part of the elicitation interface, it implements
a mechanism for extracting formal representations of requirements that were initially specified informally. We describe each of these components in some detail in this paper. The negotiation, inconsistency resolution and requirements monitoring components of the REAGENT framework do not form part of the ADREAM tool and are being implemented separately.

The agent metaphor plays a crucial role in the design and implementation of the ADREAM system. Stakeholder agents are goal-directed, reactive and autonomous, and they are able to communicate in a semantically well-founded language with other stakeholder agents, the common repository as well as human users. Our design of the ADREAM system includes several other novel proposals. The common repository includes the ability to generate complete histories of past agreements (i.e., specifications), leading to a notion of specification rationale that can be used in a manner analogous to the traditionally well-understood requirements rationale [17]. We also devise a novel approach to elicitation, based on an underlying formal definition of completeness of a specification.

The remainder of this paper is structured as follows. Section 2 presents a description of the main components of the ADREAM system. Section 3 presents a brief description of the implementation. Section 4 discusses related work.

2 ADREAM

The ADREAM tool consists of three distinct components:

- A common requirements repository (RR).
- An RE-Portal (REP). This is a web-based interface, which stakeholders utilize to communicate with the system.
- A collection of stakeholder agents (SAs), one for each stakeholder. A user logging into the REP is directed to the appropriate stakeholder agent.

2.1 The Requirements Repository

The requirements repository performs three functions:

- It stores the set of stakeholder requirements that all stakeholders have agreed to.
- It provides the computational machinery necessary for inconsistency detection and resolution, as well as for supporting stakeholder negotiation.
- It provides a deductive mechanism for answering a variety of queries that may help analysts verify conformance of a given specification with normative quality criteria.

In some instances, one can implement this framework with distinct machineries for inconsistency detection and for supporting stakeholder negotiation. In other instances, it is also possible to use the same machinery (for example, a theorem-prover) to implement both of the last two functions mentioned above. We do not discuss this computational machinery in detail in this paper, but point out that our prototype implementation uses a first-order logic theorem-prover called OTTER [14]. It is important to note that this does not entail a commitment to necessarily using first-order logic for representing requirements. The current implementation of the ADREAM tool supports requirements specification in natural language (English) and is able to support other informal or semi-formal requirements notations with minimal modification. Similarly, the notion of full (first-order) logical consistency can be replaced by alternative (potentially weaker) notion of consistency (such as consistency with respect to a limited set of rules). Once again, correspondingly different computational machineries for querying and consistency checking can be plugged in to replace the theorem-prover with relatively little effort.

We use a three-dimensional taxonomy for requirements. On the first dimension, requirements are partitioned into the sets of essential and tentative requirements. This distinction is important in inconsistency resolution and requirements negotiation, where the intent is to never violate an essential requirement (if possible) but to accept outcomes that violate tentative requirements where necessary (additional discussion for the need for such a distinction can be found in [7, 8, 9]). On the second dimension, requirements may be tagged as public or private. A public requirement can be viewed by all stakeholders, if necessary, but a private requirement is visible only to the stakeholder agent representing the stakeholder who specified this requirement and to the common requirements repository. Such a facility is useful in situations where certain business rules, processes and practices are a source of competitive advantage to an organization which would be reluctant to reveal these to other stakeholders. On the third dimension, requirements may be viewed as accepted or rejected. An accepted requirement, at any given point in time, is a requirement that all stakeholders have agreed to (via the process of negotiation). A rejected requirement is one that has been specified by one or more stakeholders but that is not included in the currently accepted set of requirements generated by the most recent round of stakeholder negotiation. The common requirements repository only stores the current set of accepted requirements. Rejected requirements are stored by the agents representing the stakeholders that originally specified them. The distinction between accepted and rejected requirements can be useful in the requirements evolution process. Prior accepted requirements can be re-labelled as rejected as a consequence of re-negotiation triggered by
a change step. These rejected requirements are never actually discarded but are retained in the agents representing the stakeholders that originally specified them, in anticipation of future re-use (these requirements are re-considered at each re-negotiation step).

We define a requirement to be a 3-tuple \( \langle D, J, T \rangle \) where:

- \( D \) is a pair \( \langle D_{NL}, D_{FORM} \rangle \) where:
  - \( D_{NL} \) denotes a description of the stakeholder’s requirement written in informal notation. In the instance of our prototype implementation, this is natural language, but this could be replaced by UML or other semi-formal notations.
  - \( D_{FORM} \) denotes a formal representation of the stakeholder’s requirement. In our current prototype the underlying formal language is sorted first-order predicate calculus, but other formal languages could also be used. We are currently extending our implementation to incorporate a language similar to the formal assertion layer in KAOS[20] (sorted first-order predicate calculus augmented with modal temporal operators).

- \( J \) is a pair \( \langle J_{NL}, J_{FORM} \rangle \) where:
  - \( J_{NL} \) denotes the rationale for the stakeholder’s requirement, represented in informal notation.
  - \( J_{FORM} \) denotes the rationale for the stakeholder’s requirement, represented in formal notation (as discussed above).

- \( T \) denotes the type of the requirement and can be one of the following:
  - \( E_{FR} \) or essential functional requirement.
  - \( T_{FR} \) or tentative functional requirement.
  - \( E_{NFR} \) or essential non-functional requirement.
  - \( T_{NFR} \) or tentative non-functional requirement.

The need for recording requirements rationale is well-recognized by the community [17]. In our instance, the rationale also plays a role in inconsistency detection. Two requirements that are mutually consistent may be nevertheless deemed inconsistent if their rationale are in conflict (i.e., the requirements were specified for contradictory reasons). The distinction between functional and non-functional requirements is important from a cognitive and semantic perspective - some additional computational justifications are described in [9].

Project-specific ontologies play a critical role in both the requirements elicitation process and the inconsistency handling/negotiation process. We take a simplified view of an ontology as a pair \( O = \langle V_{O}, R_{O} \rangle \), where the first element \( V_{O} \) is a concept vocabulary while the second \( R_{O} \) is a set of rules. These rules can be further distinguished as structural and non-structural rules. Structural rules are used for determining how the concepts in \( V_{O} \) are organized into a concept hierarchy. Non-structural rules are used for specifying other constraints and relationships (e.g., a rule which states that overdraft limits do not apply to home mortgage accounts, in a banking application). In a manner analogous to the distributed specification and evolution of a requirements specification in our framework, a project ontology can also evolve in a distributed fashion. Stakeholders may suggest updates to the ontology by adding to the concept vocabulary and/or the rules. While updates to \( V_{O} \) are relatively straightforward (the new concepts are added to the existing set), updates to \( R_{O} \) can be non-trivial, specially since these can lead to inconsistencies. Without loss of generality, we can view the elements of \( R_{O} \) as requirements, which permits us to use the same inconsistency handling/negotiation machinery to resolve conflicts. It is reasonable to assume, therefore, that all stakeholders agree on the elements of \( V_{O} \) at any given point in time, but may disagree on candidate elements of \( R_{O} \). The current accepted specification is therefore defined as a 4-tuple \( \langle V_{O}, R_{\text{accepted}}, L, PID \rangle \) where:

- \( V_{O} \) is the concept vocabulary component of the project specific ontology.
- \( R_{\text{accepted}} \) is a set of pairs of the form \( \langle \text{Req}, \text{ID} \rangle \) where \( \text{Req} \) denotes a requirement and \( \text{ID} \) identifies the stakeholders that specified \( \text{Req} \).
- \( L \) is an event log represented as a sequence of triples of the form \( \langle \text{Trig}, \text{AL}, \text{DL} \rangle \) where:
  - \( \text{Trig} \) denotes a set of triggers. A trigger is any event which may potentially lead to a modification of the current accepted specification. Triggers can be of two kinds, the addition of a requirement and the deletion of a requirement.
  - \( \text{AL} \) is the add-list which identifies the requirements that were added to the accepted specification as a consequence of the change triggered by \( \text{Trig} \).
  - \( \text{DL} \) is the delete-list which identifies the requirements that were deleted from the accepted specification as a consequence of the change triggered by \( \text{Trig} \).
- \( \text{PID} \) provides the ID of the project that the specification relates to. This permits us to handle multiple projects concurrently.

The event log \( L \) permits us to “roll-back” transactions (by removing the add-list and adding the delete-list to the current specification), enabling us to generate a complete
history of prior accepted specifications. The need for this facility was motivated by a novel way of viewing the notion of requirements rationale. Traditionally, requirements rationale are viewed as reasons/motivations for accepting individual requirements. We argue that it is equally interesting (and useful) to record specification rationale, i.e., the reasons for accepting a given specification. In our context, an explanation of why a given specification came to be regarded as the accepted specification at a given point in time can be provided by listing the prior accepted specification, the trigger that led to the requirements being re-negotiated and possibly a negotiation log (a more detailed discussion of this last element is outside the scope of this paper). The notion of specification rationale finds use in the requirements negotiation process, by enabling agents to argue for specific outcomes through reference to past precedent.

The computational machinery included in the common requirements repository supports:

- Basic queries such as when a requirement was added, deleted or modified.
- Queries on the ID of the stakeholder responsible for specifying a given requirement.
- Queries on the rationale for a given requirement or the current accepted specification.
- Resolution of inconsistencies, either by focusing analyst/stakeholder attention on minimal sources of inconsistencies or by suggesting alternative resolution outcomes. The former is achieved by generating min-conflict subsets [20, 9], which enable analysts to focus attention on portions of a specification that require "repair". The latter can be achieved by generating maximal consistent subsets [9]. In effect, the simplest stakeholder negotiation protocol would involve these maximal consistent subsets being generated in turn, and requiring stakeholders to agree on one.

2.2 RE-Portal

The RE-Portal (REP) is a web portal through which stakeholders interact with the system via interfaces such as those in Figure(1,2,3). A stakeholder logs in by supplying a stakeholder ID, password and project ID (to define the project that the current session relates to, given that multiple projects might be managed by the system concurrently). Once logged in, stakeholder interaction with the system is mediated by the corresponding stakeholder agent.

2.3 Stakeholder Agents

Stakeholder agents perform four key functions:

- They provide a stakeholder-specific requirements repository of all requirements specified by the stakeholder that it represents.
- They engage stakeholders in interactive elicitation sessions.
- They generate alerts to stakeholders whenever additional stakeholder interaction with the system is required (outside of a stakeholder-initiated session).
- They communicate with other stakeholder agents and the common requirements repository.
- They provide an interface through which a stakeholder can request to have a requirement deleted (we assume that a stakeholder can only delete a requirement specified by him/her in the first instance).

The agent contains an elicitation module. The requirement elicitation module consists of a user interface. The main function of the user interface is to assist the stakeholder in defining requirements in a format that is both practitioner-accessible and semantically well-grounded. In the case of our tool, the practitioner-accessible format that we currently support is natural language (English) while the semantically well-grounded language is first-order logic. To provide a link between these two formats, a pre-defined (and domain-specific) project specific ontology is provided.

The stakeholder-specific repository, for each stakeholder, consists of all the requirements specified by the stakeholder. These are represented as a pair \( \langle R_{\text{Private}}, \text{Preferences} \rangle \) where

- \( R_{\text{Private}} \) is a set of 3-tuples of the form \( \langle \text{Req}, S, PID \rangle \) where:
Figure 2. stakeholder-specific requirements repository

- **Req** is a stakeholder’s requirement.
- **S** denotes whether the requirement is accepted or rejected (note that this distinction does not need to be explicitly made for elements of the common requirements repository since these are by definition accepted).
- **PID** denotes the project ID relative to which the requirement has been specified.

- **Preferences** is a set of preference relations, one for each project under consideration. If **Pref<sub>PID</sub>** is a preference relation specific to project **PID** then it is defined only on the set of requirements with the same project ID. In our prototype implementation, we assume the general case where each of these preference relations are partial orders, but other preference relations could also be used (such as a total pre-order). The intention is to specify preferences on requirements from a given stakeholders perspective that would be taken into account in the inconsistency resolution/negotiation process.

2.4 Elicitation Process

The elicitation process consists of four phases:

**Initial elicitation phase:** Stakeholder agents initiate the first phase of the elicitation process by loading a pre-defined ontology specific to the project under consideration into an initial elicitation screen (as shown in Figure 3). The intent here is to enable the stakeholder to enter requirements in both natural language as well as in formal notation. The interface insists on a single sentence in first-order predicate calculus for each distinct requirement described in natural language. The granularity of each distinct requirement is left up to the user. Thus a distinct requirement could consist of a single sentence or a set of sentences. The left-hand side-panel on the screen lists constructs derived from the relevant ontology to ease the process of translating natural language requirements into a formal description. Specifically, it lists predicates, constants, connectives and quantifiers. Predicates may represent concepts/classes, relationships between class instances or attribute values. Constants are derived from class instances that have already been defined. Using the constructs defined in the side-panel, the user is expected to construct a formal assertion that is, at the very least, an abstract description of the natural language requirement. The ability to extract formal descriptions from natural language requirements (even descriptions which abstract out some amount of additional detail present in the original requirement) is critical in analyzing specifications. For instance, it has been our experience in a medium-scale case study that even formal abstractions of the original requirements can support the surfacing of key inconsistencies. For every natural language description of a requirement entered, the stakeholder is prompted for a rationale supporting the requirement. The user has the option of leaving this empty. If a rationale is entered in natural language, it must be represented formally using the same process described above.

**Ontology-driven elicitation:** Our operational notion of completeness of a specification drives the ontology-driven elicitation process. A specification is deemed to be complete if it is possible to derive the truth or falsity of all ground instance of each predicate obtained from the ontology. Once a formal representation of a specification defined in the initial phase has been obtained, a stakeholder agent identifies ground instances of predicates that the specification is currently agnostic about and presents these to the user in turn as potential cognitive triggers for the surfacing of additional requirements. Note that at the end of this process the formal abstraction of the specification may still fail to satisfy our formal notion of completeness. This is not viewed as a problem - the utility of our notion of completeness is in generating the cognitive triggers for elicitation.

**Specification-driven elicitation:** In this phase, public portions of the specifications of other stakeholders are presented to the user (the stakeholder under consideration). Once again the intention is to use these as cognitive triggers for surfacing of new requirements.

**Update-driven elicitation:** Stakeholders may update their set of requirements (contained in the stakeholder-specific repository) at any time. A stakeholder may also propose changes to the project ontology (this can happen,
for instance, when a stakeholder is unable to find an appropriate concept in the concept vocabulary \( V_O \) to describe a requirement that the stakeholder seeks to specify). Both of these types of updates may serve as triggers for additional elicitation. In both cases the update results in an email alert being sent by the stakeholder agent to the stakeholder seeking a new elicitation session.

- In the case of elicitation triggered by requirements update, the new requirement (that has been specified by a different stakeholder) is simply presented to the user/stakeholder as possible cognitive trigger for the surfacing of additional requirements. Note that the new requirements will also be communicated to the common requirements repository and tested for consistency with the current accepted specification. The new requirement may trigger a new round of stakeholder negotiation. The outcome of the negotiation may involve the new requirement being tagged as accepted and included in the new accepted specification. Alternatively, it may be tagged as rejected and retained in the stakeholder-specific repository in the agent representing the stakeholder that specified the requirement.

- In the case of elicitation triggered by ontology update, two kinds of cognitive triggers are presented to the user. In the first instance, the expanded concept vocabulary \( V_O \) leads to an elicitation session based on the completeness-testing driven elicitation strategy described above. Second, any new rules added to \( R_O \) lead to a process similar to that described for elicitation based on requirements update.

**Plan-driven elicitation:** Goal-oriented requirements engineering, as defined in frameworks such as KAOS[20], involves taking high-level user goals obtained in early-phase requirements engineering and decomposing them through AND/OR goals refinement graphs to sub-goals and eventually to downstream artefacts such as designs. AND/OR goal refinement graphs can be used to automatically generate elicitation plans. The ADREAM system implements elicitation plans within a reactive agent programming framework similar to that described in [18]. Thus, an elicitation plan consists of a goal (or trigger), a context (essentially a set of plan pre-conditions) and a body, defined as a sequence of agent actions (such as querying the user, querying other stakeholder agents etc.). A simple execution cycle seeks matches of the goal with an element of the stakeholder-specific repository. If a match is found, the context is evaluated. This typically involves seeking to derive the plan pre-conditions from the formal version of the stakeholder-specific repository. If the context is satisfied, the actions contained in the body are executed. The stakeholder agent continually monitors for both update-driven and plan-driven elicitation triggers.

**Figure 3. Elicitation Interface**

**Figure 4. Implementation Overview**

3 Implementation

A prototype has been developed and implemented to validate ADREAM. We have taken a web-based approach to development and deployment. The chosen programming language was Java and the communication medium is the World Wide Web (WWW). The resulting prototype has a small footprint and is loosely coupled with a flexible plug-in architecture able to handle future modules. The resulting prototype is compliant with Sun Personal Java standards hence can be executed on PDA’s supporting Java, allowing stakeholders to participate in the requirement engineering process even while being mobile. Hence, ADREAM is also suitable for multi-location organizations spread across the globe. This gives ADREAM an edge over other frameworks. In this section, we provide an overview of the implementation of the prototype.

There are 4 components to the ADREAM implemen-
A web server housing the web interface, an Remote Method Invocation (RMI) daemon, the REP housing the repository and an agent factory housing the agents. (Figure 4)

**Web front-end**

---

**OnRequest**

- Get agent factory stub from RMI daemon.
- Get agent stub from Agent factory.
- Use agent for requirements elicitation and management.

---

**Agent Factory**

---

**OnStartUp**

- Register with RMI daemon.

**OnRequest**

- Export stub to web interface.

---

**Repository**

---

**OnStartUp**

- Register with RMI daemon.

**OnRequest**

- Transfer data.

---

**Agent**

---

**OnStartUp**

- Get repository stub from RMI daemon.
- Load project specific ontology.
- Retrieve requirements from repository.
- Interact with stakeholder.
- Update requirement in repository.

**OnRequest**

- Inter-agent communication between stakeholder agents involves the KQML agent communication language [3].

---

4 Related Work

WinWin [12] allows various stakeholders to interact with the support system and define their individual “win” conditions. “Win” conditions are placed in a centralized repository and managed by a system engineering organization. The primary focus of this approach is methodological and tool support for actual stakeholder negotiation (as opposed to automated negotiation by agents representing stakeholders). That approach also does not involve attempts to extract formal descriptions of informal requirements, or provide support for automated elicitation. DealScribe [19] supports a sophisticated deductive machinery for querying the requirements repository. DealScribe [19] deploys HyperNews[19] and ConceptBase[19]. HyperNews provides a discussion system similar to Usenet News, but it has a World Wide Web interface. ConceptBase is a deductive database which provides a concurrent multi-user access to O-Telos objects [10]. HyperNews messages represent discussions and not requirements. Another layer of processing is required to extract the requirements out of the messages. The DealScribe approach proposes an interesting approach to requirements monitoring, but does not focus on formalizing informal requirements, or providing automated support for elicitation. Frameworks such as i* [1], KAOS [20] and CREWS [13] use agent concepts in requirements modelling and specification, but these do not relate directly to our objectives.

5 Conclusions and Future Work

In this paper, we have presented an innovative approach to distributed requirements elicitation and management. We have proposed a novel agent-mediated architecture, a set of novel requirements elicitation strategies as well as a novel mechanism for eliciting formal descriptions of informal requirements. We have also proposed new approaches to conceptualizing specification rationale and completeness with respect to an ontology. Our results have been partially validated by a medium-scale case study of a law enforcement agency’s information system.

Several classes of elicitation techniques are identified in [15]: Traditional techniques, Group elicitation, Prototyping, Model-driven techniques, Cognitive techniques and Contextual techniques. It is our belief that the elicitation process deployed in ADREAM captures flavors of most of these techniques.

As part of the REAGENT project, we are also exploring the use of semantic markup via a language such as DAML + OIL [11] to abstract out a formal description of a requirements specified in informal notation such as those in UML. This work complements the results we have obtained in the ADREAM system.

As part of the REAGENT project, we are also exploring alternative approaches to stakeholder negotiation:

1. Negotiation via argumentation. Automated negotiation is a novel approach for resolving conflicts. The minimal structure of a requirement lends itself to argumentation where arguments are a pair consisting of a premise and a conclusion. In the case of requirements, the rationale is the premise and the conclusion is the requirement. Arguments can be generated from the list of requirements.

2. Negotiation via belief merging. This approach is based on semantic accounts of how preference specifications of a society of agents can be aggregated into a combined preference relation.

---

Footnotes:

1. If the current stakeholder is not represented by agent, an agent is created and assigned to the stakeholder.

2. An agent is created if an agent does not exist.
References


