Proceedings of ACM ESEC/FSE International Workshop on Intelligent Technologies for Software Engineering WITSE 2003

A.S. d'Avila G. Spanoudakis A. Zisman

Department of Computing

City University

Technical Report Series

TR/2003/DOC/01

ISSN 1364-4009

A. S. d'Avila Garcez, G. Spanoudakis and A. Zisman

Proceedings of ACM ESEC/FSE International Workshop on Intelligent Technologies for Software Engineering WITSE03

Department of Computing

City University

Technical Report Series

TR/2003/SEG/02

ISSN 1364-4009

WITSE 2003
Workshop on Intelligent Technologies in Software Engineering

ESEC/FSE 2003
9th European Software Engineering Conference
and
11th ACM SIGSOFT Symposium on the
Foundations of Software Engineering
Helsinki, Finland
September 1-5, 2003

WITSE 2003
PROCEEDINGS

Workshop on Intelligent Technologies in Software Engineering

http://witse.soi.city.ac.uk/

September 1, 2003
Helsinki, Finland

Workshop at ESEC/FSE 2003
9th European Software Engineering Conference
and
11th ACM SIGSOFT Symposium on the
Foundations of Software Engineering

WITSE 2003
TABLE OF CONTENTS

WITSE 2003
ORGANIZING COMMITTEE

Dr. Artur Garcez is a Lecturer at the Department of Computing at City University, London. He holds an M.Eng. in Computing Engineering (honours), an M.Sc. in Computing and Systems Engineering and a Ph.D. (D.I.C.) in Computing. He is the author of a number of publications on Machine Learning, specifically on the integration of Logics and Neural Networks. His research has evolved from the theoretical foundations of Artificial Intelligent systems of Neural-Symbolic Integration to its application in Bioinformatics and Software Engineering. Dr. Garcez is an editor of the Journal of Applied Logic, Elsevier. He has served on the programme committees of international conferences and workshops. He holds a Visiting Research Fellowship at the Department of Computer Science, King's College, London. He is a member of the British Computer Society's Requirements Engineering Specialist Group. He is an author of the book "Neural-Symbolic Learning Systems: Foundations and Applications, Springer-Verlag, 2002. For more information see: http://www.soi.city.ac.uk/~aag

Dr. George Spanoudakis is a senior lecturer at the Department of Computing at City University, London. He holds a BSc (Honours) degree in Informatics, a MSc degree in Artificial Intelligence and a PhD degree in Computer Science. He has been a visiting associate professor at the Department of Computer Science at the University of Crete, and a visiting lecturer at the Department of Information Systems at the London School of Economics. He has served in the program committees of several international conferences and workshops on software engineering, having chaired workshops and conference sessions. Dr. Spanoudakis has acted as a reviewer for international scientific journals and has served as the editor of Requirenautics Quarterly, the Newsletter of the Requirements Engineering Specialist Group of the British Computer Society. He has published extensively in the area of software engineering. His current research interests include requirements specification and analysis, multi-perspective software modelling, management of inconsistencies in software documentation, and software traceability and measurement. For more information see: http://www.soi.city.ac.uk/~gespan.

Dr. Andrea Zisman is a lecturer at the Department of Computing at City University, London. She holds a PhD in Computer Science, a MSc in Applied Mathematics to Computer Science (Magna cum louver), and a B.Sc. in Computer Science (Honours). Previously, she worked as a research fellow at University College London, and as a software system consultant, developer and analyst. She has been a visiting researcher at AT&T Labs Research. Dr. Zisman has been research-active in the fields of management and automated support of distributed data and software artefacts, having an extensive number of publications. She is interested in the areas of consistency management and traceability of distributed software artefacts, validation of systems models, interoperability of distributed database systems, and web services applications, to which she has applied computational intelligence techniques in many of her approaches. She has given tutorials and commercial courses on XML in several international conferences. She has served in the program committees of international conferences and workshops, has chaired workshops and conference sessions, and has acted as a reviewer for international journals. She is member of the ACM, XML UK, and associated member of the IEE. For more information see: http://www.soi.city.ac.uk/~zisman.

WITSE 2003
PROGRAMME COMMITTEE

Artur d'Avila Garcez (City University London, UK) (*co-chair*)
Eric Dubois (Research Centre Henri-Tudor, Luxembourg)
Dimitra Giannakopoulou (NASA AMES, USA)
Robert Hall (AT&T Labs Research, USA)
Antony Hunter (University College London, UK)
Julio Leite (PUC-Rio, Brazil)
Neil Maiden (City University London, UK)
Tim Menzies (Nasa IV&V, USA)
Bashar Nuseibeh (Open University, UK)
Elisabetta di Nitto (Polimi, Italy)
Alessandra Russo (Imperial College London, UK)
Camille Salinesi - Ben Achour (Paris I - Sorbonne, France)
Ken Satoh (National Institute of Informatics, Japan)
George Spanoudakis (City University London, UK) (*co-chair*)
Andrea Zisman (City University London, UK) (*co-chair*)
Didar Zowghi (University of Technology, Sydney, Australia)

WITSE 2003
WORKSHOP INTRODUCTION

Software engineering practitioners and researchers continue to face huge challenges in the software development and maintenance of software systems. The complexity of software systems and a number of recent advances in the field of computational intelligence have been providing a fruitful integration between software engineering and intelligent technologies. In the computational intelligence field, this is particularly true in the areas of model checking, validation and verification, fuzzy logic and abductive reasoning, uncertainty management and belief based reasoning, artificial neural networks and machine learning, genetic and evolutionary computing, and case-based reasoning. In the software engineering field, this integration is seen in the areas of requirements analysis and evolution, traceability, multiple viewpoints, inconsistency management, human-computer interaction design, software risk assessment, and software verification.

The International Workshop on Intelligent Technologies for Software Engineering (WITSE'03) is intended to provide a forum for presentation and discussion of a wide range of topics related to the applicability of new intelligent technologies to software engineering problems. It aims to bring together researchers from academia and industry, and practitioners working in the areas of computational intelligence and software engineering, to discuss current state of the art, existing issues, recent developments, applications, experience reports, software tools, and future research directions of intelligent technologies applied to software engineering.

WITSE'03 is a one full day event, structured around sessions composed of a keynote talk, presentations of papers rigorously selected by the programme committee, and open round table discussions. The sessions have been organised based on papers submitted from all corners of the globe. The workshop provides an opportunity for exchanging valuable information on theoretical and practical aspects of:

- Intelligent methods of requirements analysis and evolution
- Machine learning for change management and risk assessment
- Intelligent approaches for inconsistency management of software systems
- Intelligent architectures for software evolution
- Intelligent human-computer interaction design
- Intelligent technologies for traceability management
- Intelligent techniques for software validation, verification, and testing
- Empirical studies, experience, and lessons learned on applying computational intelligence to software development

We would like to take this opportunity to thank the members of the programme committee who helped in reviewing and selecting the papers submitted to the workshop, Dr Constance Heitmeyer for her keynote talk, the authors of the submitted and accepted papers, and the ESEC/FSE 2003 workshop co-chairs Dr Cecilia Mascolo and Dr Andre van der Hoek for their assistance in the organisation of the workshop.

Helsinki, September 2003
Artur d'Avila Garcez , George Spanoudakis, Andrea Zisman

Keynote

Developing High Assurance Software Systems: On the Role of Software Tools

Constance L. Heitmeyer
Head of Software Engineering Section
Center for High Assurance Computer Systems
Naval Research Laboratory, Washington DC, USA
heitmeyer@itd.nrl.navy.mil

Abstract

Recently, researchers have developed a number of powerful, formally based software tools, such as model checkers and theorem provers, for verifying properties and for detecting property violations in both software and hardware descriptions. To date, these tools have largely been used to analyze hardware designs. However, in the future, they should have significant value in analyzing the requirements and designs of software systems, especially high assurance software systems, where compelling evidence is needed that the system satisfies critical properties, such as safety and security properties. This talk describes the many different roles that formally based software tools can play in debugging, verifying, and testing software systems and software system artefacts. It also describes one important activity in software development not involving tools that is often neglected and that merits much greater care and attention.

Abstracting Wizards from Portal Observations

Christopher J. Hogger
Imperial College London
South Kensington Campus
London SW7 2AZ, UK
+44 (0)20 75948182
cjh@doc.ic.ac.uk

Frank R. Kriwaczek
Imperial College London
South Kensington Campus
London SW7 2AZ, UK
+44 (0)20 75948447
frk@doc.ic.ac.uk

ABSTRACT
We describe a procedure for abstracting, from observations of a portal user's activities, specifications of wizard tools which, if implemented within the portal, could enable those activities to be more conducted more efficiently. The abstraction process uses a re-programmable rule-base to decide whether sequences of observed events are instances of coherent work patterns that can be used as specifications for wizard-building.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *data abstraction, patterns.*

I.5.2 [**Pattern Recognition**]: Design Methodology – *feature evaluation and selection, pattern analysis.*

General Terms
Algorithms, Design, Human Factors.

Keywords
Portal events, user actions, coherence rules, wizard tools.

1. INTRODUCTION
Various characteristics have been ascribed to an enterprise portal [1, 2, 3]. In broad terms such a portal provides access to resources in a manner customized to its users.

Our work addresses one particular approach to such customization. The typical user instigates many information-processing events: opening emails and attachments, creating documents, updating databases, and so on. In this stream of activity there are many repetitive sequences, arbitrarily interleaved in time, each constituting a coherent set of events. If enough such sequences conform to some abstract template then that template can be used to build a generic tool potentially helpful to the user.

Automating wizard building has been examined elsewhere [4, 5] but within a more limited context and with a narrower focus.

Our focus on recognition of event sequences, based upon that of [6, 7], is somewhat related to [8] in which an algorithm gradually adapts a user interface to its observed pattern of use.

Related work in active databases and workflow includes concatenating recurring sequences of primitive events. An active database monitors transactions to develop production rules for such tasks as integrity enforcement. As some rules may take effect only after particular combinations of events have occurred, it is important to specify composite events flexibly [9, 10], as well as to detect them efficiently [11].

Workflow is concerned with the coordination of uses, information and events that cause work to flow, such as deadlines, and the status of work required to meet organisational objectives. Workflow researchers are interested in reusing business process models, including templates (models that have been especially adapted for reuse) [12]. Such templates are essentially complex composite actions. In [13], data mining is used to construct process models from workflow logs, thereby creating new workflow systems conforming to previous behaviour.

2. OUTLINE OF THE PROCEDURE
We focus on portal evolution instigated by observations of events, namely user actions. The portal is presumed to contain a background agent that observes, stores and analyses these events. The agent is encoded as a logic program, partly for ease in codifying rule-based principles but mainly in order to manipulate partially-determined data structures. Its role is to trigger appropriate portal responses to the events observed. We restrict the response to action composition, characteristic of wizard-building and toolbar customization, whose simplest mode is concatenation. This arranges selected actions into ordered sequences representing composite actions, whose generic form specifies the desired wizard.

To analyse events we require access to the portal's event history to ascertain their temporal order and frequencies of occurrence. Each event updates this history. For practicality, policies are needed to govern the granularity, relevance and temporal baseline of the events recorded in the history.

Whenever a new event arrives in the history, our procedure analyses the sequence — named Events — comprising the most recent events as determined by a given bound upon the times since they entered the history. The analysis decides whether Events includes an instance of a coherent behavioural pattern and, if so, whether that pattern has been observed sufficiently frequently. If these properties hold then the procedure signals that the pattern is worth converting (offline) into a tool for subsequent integration into the portal's functions.

3. CHARACTERIZING EVENTS

The characterization of an event determines what can be done in response to it. We represent an event as a term of the form event(U, T, A, R, Ts) whose arguments are:

user (U)	*the initiator of the event*
tool (T)	*the tool employed*
action (A)	*the action applied using the tool*
resources (R)	*the resources so entailed in the action*
timestamp (Ts)	*the time when the event was completed*

An event history is a set of such terms. The representation imposes some degree of granularity upon the observable events. For instance, we do not observe what happens between the initiation and completion of an event. Further, we focus mainly upon events whose actions apply tools to resources.

Deciding suitable types for the arguments is a matter of practical choice. Any such choice must be sufficiently general to match realistically observable events, but sufficiently specific to enable realistically useful responses to flow from them. Our own choices have been applied so far in one medium-scale, real-world context to test their suitability. Later we will briefly illustrate an example drawn from that exercise.

Below we outline the types so far experimented with. They are not a fixture within the general framework, and can easily be varied to suit the portal domain.

3.1 User (U) Types

We take only the simplest case in which U is an atom serving as a user identifier. In most cases U identifies the portal's user. In some cases, however, it may identify a different user, enabling the portal to observe interactions between users.

3.2 Tool (T) Types

A term tool(Name, Exec, Location, Rights, Params) is used to represent a tool T. All five arguments are represented by atoms whose interpretations are as follows:

Name: the internal system name for the tool, appropriate to the host operating system. In Windows, for example, 'MSWord' could be such a name. A tool name in Unix might be 'emacs'.

Exec: the internal system name for the executable file representing the tool. It may take the form *Filename.ext* in cases where the file has an associated extension.

Location: the location of the executable file. This could be, for instance, a path on a local drive—such as '/etc/bin/emacs/'—or a URL to a remote computer.

Rights: a member of {true, false} denoting whether or not the user has the right to access the executable file.

Params: the values of any parameters the tool expects to be supplied by the user when invoking the tool. If none are needed then Params takes the value null.

3.3 Action (A) Types

The framework currently supports a variety of conceptually distinct kinds of user action, denoted by the following atoms:

created	copied	deleted	logged_in
moved	pasted	sent	logged_out
renamed	received	opened	updated

Thus, a term event(U, T, created, R, Ts) signifies an event in which the user U used tool T to create resource R at time Ts.

An action A does not require arguments of its own. All the material upon which it bears is contained in the other top-level arguments—primarily T and R —of the event term.

Not all actions involve tools or resources. Actions which do must be compatible with them. Implicitly, there are further constraints upon the type characterizations of events in order to render them meaningful. For instance, the constraints upon event(U, T, logged_in, R, Ts) require U to be the portal's user, T to be the atom null (no tool is employed in logging-in), R to be null (no resources are involved) and Ts to be not null. By contrast, the constraints upon event(U, T, created, R, Ts) require U to be the portal's user, T to be not null, R to comprise precisely one resource and Ts to be not null. A suitable set of assumed constraints for this domain can be found in [6].

When a null atom appears in an argument position within an event term, it signifies that the argument is *undefined*, that is, has no concrete meaningful value. This provision must be clearly distinguished from that which permits an argument to be *undetermined*. In the latter case the argument is an anonymous variable ('_') . So a term event(U, _, moved, R, Ts) represents an event in which resources R were moved (between locations) but in which the tool used (if any) was not observed, or not recorded or not of interest. This partially-determined structure stands generically for all the concrete terms obtainable by choosing particular values for the tool argument.

3.4 Resource (R) Types

Deciding upon suitable resource types is perhaps the hardest aspect of characterizing events. The real world offers many such types, each having its own particular kind of content, handles, and so on. We have experimented with just three primary resource types—document, database table and email, denoted by the atoms doc, dbtable and email respectively. Each one has the following attributes:

Type: its type identifier, chosen from {doc, dbtable, email }

Name: its current internal system name (an atom).

Loc: its current location, such as a local pathname or a URL (an atom).

Rights: its rights, being some subset of {read, write}.

Ucr: the identifier of the user who created it (an atom).

Tcr: either null or the local pathname of the tool used to create it (an atom).

OldN: for the action renamed this is the resource's prior name (an atom), but for all other action types it is null.

OldL: for the action moved this is the resource's prior location, being of the same type as Loc (an atom), but for all other action types it is null.

Atts: further attributes specific to the primary type: for a document it is null; for a database table it is a term of the form atts(Spec, Schema) where Spec is an atom encoding a query or view definition, and Schema is an atom encoding the table's schema; for an email it is a term atts(Sender, Recipients, Subject, Attachment) where Sender is a user identifier (an atom),

Recipients is a list of user identifiers, Subject is the message's subject line (an atom) and Attachment is a list of atoms each encoding the local pathname or URL of an attached file.

More: a place-holder for any other information about the resource that the portal might be required to exploit.

The representation of a resource is then a term containing all these arguments, as follows:

```
resource(Type, Name, Loc, Rights, Ucr, Tcr, OldN, OldL, Atts, More)
```

3.5 Timestamp (Ts) Types

A timestamp is represented by a term whose arguments fix the temporal granularity for recording events. We use a term ts(Year, Month, Day, Hour, Min, Sec). Whenever an event's action is completed, the arguments of its timestamp are instantiated with values drawn from the host system clock.

A complete example of an event drawn from the real-world data we have experimented with is the following:

```
event('u146', tool('Outlook', 'outlook.exe',
  '/ProgramFiles/Outlook', true, null), opened,
  [resource(doc, 'agenda.agd' , '/users/u146/Outlook/',
  [read, write], u146,
  'ProgramFiles/Outlook/outlook.exe', null, null, null, null)],
  ts(2000, 9, 5, 9, 10, 44))
```

This represents the completion at time 9.10.44 a.m. on September 5th 2000 of the action by user u146 of opening with the Outlook tool a single resource, namely a document 'agenda.agd' — drawn from the local directory /users/u146/Outlook/ — which that same user had previously created with that same tool.

4. EVENTS AND WORK PATTERNS

Events are real and tangible manifestations of an individual's work. We can identify them not only correctly but also unobtrusively. More difficult is the task of deciding whether they jointly signify meaningful work patterns. A sequence of temporally-ordered events may be recognizable as a whole to the user in relation to their defined role. However, the flexibility of working arrangements to achieve overall objectives may be such that some sequences, whilst logically consistent with the role, may not be directly recognizable as such. Moreover, the environment may be one in which no individual roles have been formally defined.

Given this, we employ a customizable rule-base to decide whether any given event sequence has a subsequence matching some work pattern meriting wizard support in the portal. This allows flexibility in determining such patterns without consulting the user or institutional roles, although it does not preclude doing so. The rule-base recognizes such patterns in the event history on the basis of the events' internal content.

The event history is conceptually a set of stored events. In general, however, events have a natural temporal order determined by their timestamps. In the absence of such an order a set of events may not amount to a set of logically coherent actions. For example, a resource cannot be sent as an email attachment before it has been created. A coherent set will normally require that its timestamps satisfy at least some given partial order, and in some cases may even need to be totally ordered. Given two particular events e1 and e2 we might require that e1 occurred before e2 and/or that the interval between them were less than some specified bound.

An event-collecting agent running in the portal background produces the event history as a stream of validated event terms in the form described above. An adjustable bound controls how long events persist in the history before being garbage-collected. Concurrently, a wizard-analysing agent eagerly consumes the history's events. Each newly-arrived event triggers this analyser to assemble the more recent events — including the one just arrived — into a list called Events, and then to seek significant patterns within this list by consulting the rule-base. For coding convenience, Events is ordered by increasing recency. Its membership is constrained by another adjustable bound that determines how far back through the event history the search for patterns is pursued.

For each instance of Events the analyser potentially considers each non-empty subsequence S within it. S need not be contiguous. For instance, if Events has the form [e1, e2, e3, e4, e5, e6, ...] then S might be [e1, e3, e4, e6]. Whether S is an instance of a work pattern depends upon it being recognized as such by *coherence rules* in the rule-base. These customizable rules describe work-related relationships among events.

In general, the notion of coherence of a subsequence can be based upon any logical relation over its members. In this paper we consider coherence only between its *adjacent* members. This reduces the discernibility of potential patterns but gains from the simplicity of defining pairwise constraints. Given this, each coherence rule takes the form

```
cohere(C, Cnext) :- conditions.
```

A coherent subsequence [e1, e3, e4, e6] must then satisfy cohere(e1, e3), cohere(e3, e4) and cohere(e4, e6). It is not, however, the structure we wish ultimately to extract. In a coherence rule the head arguments C and Cnext are typically non-ground, that is, contain variables. When this is so, C and Cnext are required to cohere for whatever values they take. So, it may be enough that C has an opened action and Cnext a copied action, without regard to the particular tools employed in those actions. In that case the tool arguments within C and Cnext would be anonymous variables. Moreover, their anonymity would prevent them from being bound by the evaluation of the rule's conditions. What we wish to extract is not the concrete subsequence, but rather an abstract template for it that has no greater specificity than that required by the coherence rules. For [e1, e3, e4, e6], one extracted template may be [t1, t3, t4, t6]. Then, e1, ..., e4 must be substitution instances of t1, ..., t4 and the rule-base must contain satisfied rules whose heads are cohere(t1, t3), cohere(t3, t4) and cohere(t4, t6).

The logic program shown below extracts all templates from Events. The principal case is the final scan clause, whose logic is such that a partially-constructed template T (in reverse temporal order) is extended by a term Cnext provided (a) a pair (E, Enext) of concrete events satisfies some coherence rule whose head's argument tuple is a variant copy of (C, Cnext) and (b) C unifies with the term Cprev most recently put in T. Requirement (b) ensures that each abstract term in the template coheres with both its predecessor and its successor, as is identically required of the corresponding concrete terms in the input subsequence. The final state of T is reversed by the first scan clause to yield a completed template in order of increasing recency.

```
find_templates(Events, Templates) :-
  findall(Templ, template(Events, Templ), Templates).

template(Events, Templ) :-
  scan(Events, [], [], Templ), length(Templ, N), range(N, 3, 10).

scan([], _, T, Templ) :- reverse(T, Templ).
scan([_ | Es], B, T, Templ) :- scan(Es, B, T, Templ).
scan([E | Es], [], T, Templ) :- !, scan(Es, [E], T, Templ).
scan([Enext | Es], [E], [],Templ) :-
  cohering_pair(E, Enext, C, Cnext), !,
  scan(Es, [Enext], [Cnext, C], Templ).
scan([Enext | Es], [E], [Cprev | T], Templ) :-
  cohering_pair(E, Enext, C, Cnext), C=Cprev, !,
  scan(Es, [Enext], [Cnext, Cprev | T], Templ).

cohering_pair(E, Enext, C, Cnext) :-
  clause(cohere(A, Anext), Conditions),
  copy_term((A, Anext), (C, Cnext)), E=A, Enext=Anext,
  Conditions.
```

One can improve the procedure so that when it is triggered it need not test for coherence of any pair previously tested. This can be arranged by indexing the events in the history and consulting a database of pairs already shown to cohere.

A recognized template is required to have a sensible length. The program above delivers only templates having lengths in the range 3-10. A length shorter than 3 represents so little activity that a corresponding wizard would confer little realistic benefit, whilst for one above 10 the wizard could be overly specialized or unwieldy to use.

5. COHERENCE RULES

The following is an example of a coherence rule.

```
cohere(  event(U, _, received, Rs, Ts1),
         event(U, _, opened,  [R], Ts2) :-
    portal_user(U), U\==Sender,
    R=resource(  email, _, _, _, _, _, _, _,
                 atts(Sender, _, _, _), _),
    member(R, Rs), soon_afterwards(Ts1, Ts2).
```

It identifies coherence between the receipt by the portal's user U of a collection Rs of resources including an email R which U opens "soon afterwards", conditional upon R having been sent by someone other than U. The additional requirement that U shall be a recipient of the email will have been already checked during the stream-entry validation of whichever event is now matched with the rule-head's first argument.

An example of a concrete Events list containing at least two events is:

```
[event(u146, null, received,
  [resource(email, 'inbox.dox',
   '/Win/ApplicationData/Identities/u146/Outlook/',
  [read, write], null, null, null, null, atts(u416, u146,
   'Book Meeting', null), null), resource(email, 'inbox.dox',
   '/Win/ApplicationData/Identities/u146/Outlook/',
  [read, write], null, null, null, null, atts(u811, u146,
   'Request', '~pn105'), null)], ts(2000, 9, 5, 9, 2, 52)),
 event(u146, tool(Outlook, outlook.exe,
   '/ProgramFiles/Outlook', true, null), opened,
  [resource(email, 'inbox.dox',
   '/Win/ApplicationData/Identities/u146/Outlook/',
  [read, write], null, null, null, null, atts(u416, u146,
  'Book Meeting', null), null)], ts(2000, 9, 5, 9, 10, 20)),
   ... and further events ...]
```

This data was extracted from the actions observed over a period of a few weeks in the "real life" of a business portal user. The event history accumulated amounted to several thousand events. The first event above records that user u146 received two emails, one from u416 about the booking of a meeting and the other from user u811 issuing a request. The second event records that about seven minutes later u146 opened the first of those emails. Provided that this elapse time is within the period defined as "soon afterwards", the coherence of these events as defined by the rule above yields a template for u146's portal having the form

```
[event(U, _, received, _, _),
 event(U, _, opened, [_], _), ... and further events ...]
```

This reflects no more than the pattern of receiving resources and then opening a single resource, not necessarily among those received. A wizard constructed from it may support just that level of abstraction and enforce nothing further. This depends upon the process used to construct wizardss. If the coherence rule is made more specific:

```
cohere(event(U, _, received,
      [resource(email, N, L, _, _, _, _, _, Atts, _)], Ts1),
      event(U, _, opened,
      [resource(email, N, L, _, _, _, _, _, Atts, _)], Ts2) :-
      portal_user(U), U\==Sender,
      Atts=atts(Sender, _, _, _), soon_afterwards(Ts1, Ts2).
```

we obtain a more specific template reflecting the receipt of just one resource, necessarily comprising an email, and the subsequent opening of that same email:

```
[event(U, _, received,
  [resource(email, N, L, _, _, _, _, _, Atts, _)], _),
 event(U, _, opened,
  [resource(email, N, L, _, _, _, _, _, Atts, _)], _),
 ... and further events...]
```

Figure 1 illustrates a simple wizard built from this template.

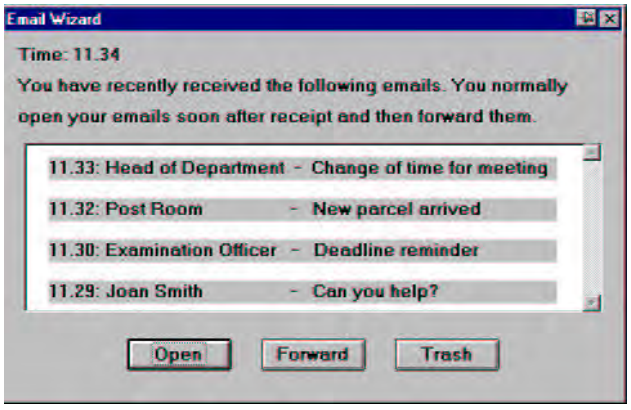

Figure 1. An email-handling wizard.

This is launched in the portal interface when an unopened email passes the "soon afterwards" threshold. Its main features can be built mechanically by a template interpreter. The window of emails waiting to be opened, and its "Open" button, are such features. The "Forward" button is owed to a third event in the template recording that user U, upon opening the email, executes a sent action upon it to forward it

elsewhere. The text field "You normally open your emails soon after ..." is devised manually from inspection of the conditions in the body of the coherence rule. The "Trash" button is an embellishment added without recourse to either the template or the coherence rule.

The expressive power of the templates is limited to what is expressed solely by the arguments appearing in their abstract event terms. Conditions in the bodies of the coherence rules cannot form part of the templates. In the above example, the condition that the email's sender shall not be the portal's user is not enforced by the template. Likewise, we cannot express in a template that the opened email must belong to some longer list of received resources, nor that it be opened "soon afterwards". More generally, however, wizard-building can take account of conditions in the coherence rules' bodies, either using them to constrain the construction process or actually compiling them into the wizards.

The above treatment identifies templates using just the coherence rules, taking no account of the frequency with which events occur. It treats a pattern as worth instituting as a wizard even if it is observed to occur only once. More rational is to require that a pattern p shall also have a sufficiently high frequency of occurrence over some allotted time period, in relation to the frequencies of occurrence of its constituent events. Our framework currently employs only a simple measure of this, using the frequency fraction $fp / (f1 + ... + fn)$ where n (>1) is the length of p, fp is the number of occurrences so far of p and each fi is the number of occurrences so far of event ei (ignoring its timestamp). The nearer this fraction approaches 1, the more significant is p within the event history. For each n we stipulate a threshold tn above which the fraction is taken to indicate that p justifies building a wizard.

6. CONCLUSION

Wizard tools can offer useful economies to the user provided their construction is unobtrusive and provided they capture significant work patterns. In this work we have focused upon the role of logical characterization of work patterns. This aspect can be integrated with other capabilities for the data-mining of event histories and for the incremental adaptation of wizards. Data-mining might also assist the development of the coherence rule-base. However, the rule-base alone, with limited frequentist filtering, does provide significant power in identifying template candidates, and many examples more interesting—but too lengthy to illustrate here—than the above example are derivable from our experimental data.

Other researchers have applied machine learning and other data mining techniques to analyse sequences of user actions in order to build models that attempt to anticipate user needs and hence reduce the burden of working with a computer. In [14] graph-based induction — an information-measure based technique for pattern detection — is used in three distinct ways: (i) to monitor the input of user commands in real-time and constantly offer the "next command" as anticipated by the model, (ii) to produce stereotypical scripts composed of short sequences of commands that have been observed as occurring frequently through the batch processing of long histories of operations and (iii) to produce rules — also through batch processing — that can be used to prefetch files in a multi-tasking environment. The approach we describe in this paper is closest in intent to the second of these tasks. However, our wizards can be quite flexible tools, whereas the scripts described in [14] are fixed sequences allowing only for single arguments. A degree of expertise is required in the statement of the coherence rules upon which our wizards are constructed whereas the scripts in [14] are produced by an automated machine learning process. Yet expertise is still required to weed out those scripts deemed to have little utility.

7. REFERENCES

[1] Firestone, J.M. Defining the Enterprise Information Portal. Executive Information Systems, Inc., 1999.

[2] Wells, D., Sheina, M. and Harris-Jones, C. Enterprise Portals: New Strategies for Information Delivery. Ovum Report, 2000.

[3] Murray, G. The Portal is the Desktop. Group Computing. May-June 1999, p.22.

[4] Carliner, S. Designing wizards. Training and Development, 1998.

[5] Kipp, N.A. Hyperwizards: XML over CGI. Proc. of XML'98, 1998.

[6] Serafetinidou, M. Wizard Building in an Abstract Portal Framework. MSc Dissertation, Department of Computing, Imperial College London, 2000.

[7] Hogger, C.J., Kriwaczek, F.R. and Serafetinidou, M. Wizard Building in an Abstract Portal Framework. Technical Report 2000/18, ISSN 1469-4174, Department of Computing, Imperial College London, 2000.

[8] Davison, B.D. and Hirsh, H. Predicting Sequences of User Actions. Proc. of the 1998 AAAI/ICML Workshop Predicting the Future: AI Approaches to Time-Series Analysis, 1998.

[9] Motakis, I. and Zaniolo, C. Composite Temporal Events in Active Database Rules: A Logic-Oriented Approach. Proc. of the 4th Intl. Conference on Deductive and Object-Oriented Databases, 1995.

[10] Motakis, I. and Zaniolo, C. Temporal aggregation in active databases rules. SIGMOD Rec. 26(2), pp.440-451, 1997.

[11] Urban, S.D., Unruh, A., Martin, G. and Nodine, M. Expressing Composite Events in InfoSleuth. Microelectronics and Computer Technology Corporation, Technical Report #MCC-INSL-131-98, 1998.

[12] Jørgensen, H. and Carlsen, S. Writings in Process Knowledge Management: Management of Knowledge Captured by Process Models. ISBN 82-14-01928-1. SINTEF Telecom and Informatics, 2000.

[13] Agrawal, R., Gunopulos, D. and Leymann, F. Mining Process Models from Workflow Logs. Proc. of the Sixth International Conference on Extending Database Technology (EDBT), 1998.

[14] Motoda, H. and Yoshida, K. Machine learning techniques to make computers easier to use. Artificial Intelligence 103, pp.295-321, 1998.

Computing Minimal Revised Specifications by Default Logic

Ken Satoh
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ksatoh@nii.ac.jp

ABSTRACT

This paper presents a method of computing minimal revised specifications represented as a function-free Horn theory. We have already proposed a formalization of minimal revision of a logical specification and two computational methods. However, the previous methods have either a problem of needs of minimality check or a problem of completeness. In this paper, we propose a method using Default Logic which not only directly computes a minimal revised specification without minimality check, but also is guaranteed to be complete. Then, we give a top-down proof procedure to compute an extension correpsonding with a minimal revused specification.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Methodologies*

1. INTRODUCTION

Software evolution is one of the most important issue in software engineering. The famous report [8] even in 80's states that 75% of maintenance task in software industry are activities dealing with the following issues.

- Changes in the software environment.
- New user requirements.

In the current network environment of computers, the above correction/updates occurs frequently and therefore the research on software evolution becomes much more important.

As the first steps, we have already formalized an update for a specification represented as a set of Horn theories with integrity constraints. We divide a specification into two parts; a temporary part and a persistent part. The former is subject to change and the latter is not allowed to be changed. Then, when the addition of a new specification leads to contradiction, we *minimally* revise a logical specification to avoid contradiction. So far, we proposed the following methods to compute minimal revised specifications.

Using abductive logic programming [16, 17] We translate a logical specification into an abductive logic program where we introduce a deletion abducible into each rule of the temporary part of the logical specification and correspond the assumption of deletion abducible with a deletion of the temporary part.

Using extended logic programming [18] We translate a logical specification into an extended logic program where we consider all the contrapositive of the logical specification in order to simulate a logical deduction within extended logic programming and we introduce a deletion literal into each rule of the temporary part of the logical specification and correspond the truth of deletion literals with a deletion of the temporary part.

However, either method has its own problem of computing minimal revised specification. The former method firstly computes a set of deletion abducibles corresponding with the deletion of rules to avoid contradiction, but then we need to check its minimality since the computing method in [16, 17] is not guaranteed to compute minimal abducibles. On the other hand, the latter method does not need to perform minimality check, but the method is not complete for every class of logical specification; in other words, there is a minimal revised specification, but by the latter method, we cannot find it. So, we have to restrict the class of logical specification to guarantee completeness.

In this paper, to solve these problems, we propose a usage of Default Logic [12]. We translate each logical specification into a default theory and compute an extension of the theory. Then, we can guarantee that each extension exactly corresponds with a possible minimal revised specification. We also give a procedure of Default Logic which is simplied from our original procedure of Default Logic [15] to compute (a part of) an extension to identify which rules should be deleted.

2. MINIMAL REVISED SPECIFICATION

In this section we review our framework of *minimal revised specification*[16, 17, 18] for maintaining consistency of software specification.

Consider the following example of a logical representation of database and constraints which is inspired by the example in [4, p590]. Here, we assume that data in database, integrity constraints, and rules are represented as logical formulas.

Example 1.

- Integrity Constraint meaning that if F is a father of E then the age of E must be under the age of F:
  $father(F, E) \wedge age(F, A1) \wedge age(E, A2) \wedge (A1 \leq A2) \supset \bot.$
  where $\bot$ means "contradiction".
- A rule of calculating the age:
  $birth\_year(E, Y) \wedge current\_year(Z) \wedge A{=}Z - Y \supset age(E, A).$
- s is the father of c:
  $father(s, c)$.
- The birth year of s is 1975:
  $birth\_year(s, 75)$.
- The birth year of c is 1974:
  $birth\_year(c, 74)$.

We assume that the following:

- The integrity constraint is not completely specified. There would be an exception of the above integrity constraint.
- There are some possibilities that the information of birth year is wrongly inserted in the database.
- Therefore, we can delete either some part of the integrity constraint or some date of birth year to avoid future contradiction.

Suppose that we add $c(99)$.[1]. The addition of $c(99)$. leads to inconsistent database state. To resolve such inconsistency, we would consider the following possibilities.

S_1**:** $b(s, 75)$ is considered to be added incorrectly and so, we delete this information from the database.

S_2**:** $b(c, 74)$ is considered to be added incorrectly and so, we delete this information from the database.

S_3**:** We regard this situation as an exceptional situation and so, we change the integrity constraint.

Note that there are other ways of resolving inconsistency, but other revisions are greater than the above three revisions. In this paper, we would like to have such minimal revised specifications.

[1]From now on, for brevity, we write "f" for "$father$" predicate, "a" for "age" "b" for $birth\_year$, "c" for $current\_year$, respectively

Definition 1. Let T_{pst} be a set of Horn clauses which are of the form:

$$B_1 \wedge B_2 \wedge ... \wedge B_l \supset H.$$

where H, $B_1, ..., B_l$ are atoms.

Let T_{tmp} be a set of labeled Horn clauses which are of the form:

$$\phi : \ B_1 \wedge B_2 \wedge ... \wedge B_l \supset H$$

where ϕ is a name for the clause.

A *logical specification* T is a pair of $\langle T_{pst}, T_{tmp} \rangle$ and we call T_{pst} a *persistent part* of T and T_{tmp} a *temporary part* of T.

We define a minimal revised specification based on a maximal consistent subset of the logical specification defined as follows.

Definition 2. Let S be a set of function-free Horn clauses. A *maximal consistent subset* of S is S' such that S' is consistent and $S' \subseteq S$ and there is no proper superset S'' of S' such that S'' is consistent and $S'' \subseteq S$.

Definition 3. Let T be a logical specification $\langle T_{pst}, T_{tmp} \rangle$. Let $\Pi_{T_{tmp}}$ be a set of ground clauses obtained by replacing all variables in each clause in T_{tmp} by every constant in T. Let R_{new} be a clause.

A *minimal revised specification* w.r.t. T and R_{new} is $\langle T_{pst} \cup \{R_{new}\}, S \rangle$ such that $(T_{pst} \cup \{R_{new}\}) \cup S$ is a maximal consistent subset of $(T_{pst} \cup \{R_{new}\}) \cup \Pi_{T_{tmp}}$.

The minimal revised specification is such a specification that deletes ground instances of T_{tmp} when $T_{pst} \cup \{R_{new}\} \cup T_{tmp}$ leads to contradiction.

Example 2. Consider the specification in Example 1.

$T_{pst} \cup \{R_{new}\}$:
$b(E, Y) \wedge c(Z) \wedge A = Z - Y \supset a(E, A)$.
$f(s, c)$.
$c(99)$

T_{tmp} :
$b(s, 75)$.
$b(c, 74)$.
$f(F, E) \wedge a(F, A1) \wedge a(E, A2) \wedge (A1 \leq A2) \supset \bot$.

$\Pi_{T_{tmp}}$is as follows.
$b(s, 75)$.
$b(c, 74)$.
$f(s, c) \wedge a(s, 24) \wedge a(c, 25) \wedge (24 \leq 25) \supset \bot$.
$f(s, c) \wedge a(s, 25) \wedge a(c, 25) \wedge (25 \leq 25) \supset \bot$.
$f(s, s) \wedge a(s, 24) \wedge a(s, 24) \wedge (24 \leq 24) \supset \bot$.
....

Since $T_{pst} \cup \{R_{new}\} \cup \Pi_{T_{tmp}}$ is contradictory, we have the following minimal revised specifications $\langle T_{pst} \cup \{R_{new}\} and S_i \rangle (i = 1, 2, 3)$.

- $S_1 = \Pi_{T_{tmp}} - \{b(s, 75).\}$.
- $S_2 = \Pi_{T_{tmp}} - \{b(c, 74).\}$.
- $S_3 = \Pi_{T_{tmp}} - \{f(s, c) \wedge a(s, 24) \wedge a(c, 25) \wedge (24 \leq 25) \supset \bot.\}$.

3. COMPUTING MINIMAL REVISED SPECIFICATION BY DEFAULT LOGIC

In this section, we give a translation from a logical specification to a default theory in order to compute a minimal revised specification.

Definition 4. Let T be a logical specification $\langle T_{pst}, T_{tmp} \rangle$ where T_{pst} and T_{tmp} are function-free. We define *a translated default theory for a consistency management of* T (denoted as $DT_{CM}(T) = (D, W)$) as follows.

- W consists of the following clauses.
  - Every clause in T_{pst} is in W.
  - Let $\phi : \ B_1 \wedge ... \wedge B_l \supset H \in T_{tmp}$. We introduce a *deletion literal* $del_\phi(\mathbf{x})$ for each clause ϕ where $\mathbf{x}$ is a tuple of free variables in ϕ. Then, a clause $B_1, ..., B_l \supset H \vee del_\phi(\mathbf{x})$. is in W.
  - Unique name axioms and domain closure axioms are in W.
- D consists of the following defaults.

$$\{\frac{: \neg del_\phi(\mathbf{x})}{\neg del_\phi(\mathbf{x})} | (\phi : C(\mathbf{x})) \in T_{tmp}\}$$

Definition 5. Let $\phi : B_1 \wedge B_2 \wedge ... \wedge B_l \supset H$ be a labelled clause and $\mathbf{X} = \langle X_1, ..., X_k \rangle$ be a tuple of free variables in ϕ and $\theta_1, ..., \theta_m$ be substitutions of ground terms to these free variables. *Updated clause* $Update(\phi; \theta_1, ...\theta_m)$ *of* ϕ *w.r.t.* $\theta_1, ..., \theta_m$ is defined as the following clause:

$$\phi' : \neg EQ(\mathbf{X}, \theta_1) \wedge ... \wedge \neg EQ(\mathbf{X}, \theta_m) \wedge L_1 \wedge ... \wedge L_l \supset H$$

where ϕ' is the new name of the above clause and $EQ(\mathbf{X}, \theta_i) = ((X_1 = (X_1\theta_i)) \wedge ...(X_k = (X_k\theta_i)))$

The following theorem shows that an extension of $DT_{CM}(T)$ exactly corresponds with a minimal revised specification.

Theorem 1. Let T be a function-free logical specification $\langle T_{pst}, T_{tmp} \rangle$, and let R_{new} be an added clause. If there is an extension E for $DT_{CM}(\langle T_{pst} \cup \{R_{new}\}, T_{tmp} \rangle)$ with a set of deletion literals Δ, then $\langle T_{pst} \cup \{R_{new}\}, (T_{tmp} - T_{del}) \cup T_{add} \rangle$ is a minimal revised specification where

$T_{del} = \{\phi : \ B_1, ..., B_l \supset H | (del_\phi(\mathbf{x})\theta) \in \Delta\}$ and

$T_{new} = \{Update(\phi; \theta_1, ...\theta_m) | (del_\phi(\mathbf{x})\theta_i) \in \Delta\}$.

Conversely, if there is a minimal revised specification $\langle T_{pst} \cup \{R_{new}\}, T_{new} \rangle$, then there exists an extension E for $DT_{CM}(\langle T_{pst} \cup \{R_{new}\}, T_{tmp} \rangle)$ s.t. $\Pi_{T_{tmp}} - \Pi_{T_{new}} = \{\phi : (B_1, ..., B_l \supset H)\theta | (del_\phi(\mathbf{x})\theta) \in E\}$.

PROOF. We use the following proposition mentioned in[13]. Let D be a set of defaults, each of the form $\frac{: \beta}{\beta}$. We denote a set of consequents of defaults, $\{\beta | \frac{: \beta}{\beta} \in D\}$ as $CONS(D)$.

Proposition 1. Suppose R is a set of default rules, each of the form $\frac{: \alpha}{\alpha}$ for a first-order sentence, α. Then E is an extension for the default theory (R, W) iff $E = Th(W \cup CONS(D))$ where D is a maximal subset of R such that $W \cup CONS(D)$ is consistent.

Proof of Theorem 1: By the above proposition and the definiinition of $DT_{CM}(T)(= \langle D, W \rangle)$, E is an extension of $DT_{CM}(T)$ if and only if $W \cup \{\neg del_\phi(\mathbf{t}) | \neg del_\phi(\mathbf{t}) \in E\}$ is a maximal consistent subset of $W \cup \{\neg del_\phi(\mathbf{t}) | \phi \in T_{tmp}\}$. This equivalently means that $T_{pst} \cup \{R_{new}\} \cup (T_{tmp} - T_{del}) \cup T_{add}$ is a maximal consistent subset of $(T_{pst} \cup \{R_{new}\}) \cup \Pi_{T_{tmp}}$, or equivalently, $\langle T_{pst} \cup \{R_{new}\}, (T_{tmp} - T_{del}) \cup T_{add} \rangle$ is a minimal revised specification. □

Example 3. Consider the database specification $T = \langle T_{pst}, T_{tmp} \rangle$ in the Example 1. To compute a minimal revised specification, we translate the specification into the following default theorm (D, W).

W:
$b(E, Y) \wedge c(Z) \wedge A = Z - Y \supset a(E, A)$.
$f(s, c)$.
$c(99)$
$b(s, 75) \vee del_{\phi_1}$.
$b(c, 74) \vee del_{\phi_2}$.
$f(F, E) \wedge a(F, A1) \wedge a(E, A2) \wedge (A1 \leq A2) \supset del_{\phi_3}(F, E, A1, A2)$.

D:

$$\frac{: \neg del_{\phi_1}}{\neg del_{\phi_1}}, \quad \frac{: \neg del_{\phi_2}}{\neg del_{\phi_2}}, \quad \frac{: \neg del_{\phi_3}(F, E, A1, A2)}{\neg del_{\phi_3}(F, E, A1, A2)}$$

Then, three extensions $E_1, E_2, and E_3$ containing the following literals for the above default theory.

$E_1 \supseteq \{del_{\phi_1}, \neg del_{\phi_2}, \neg del_{\phi_3}(s, c, 24, 25), \neg b(s, 75), \neg a(s, 24), a(c, 25), b(c, 74), f(s, c), c(99)\}$

$E_2 \supseteq \{\neg del_{\phi_1}, del_{\phi_2}, \neg del_{\phi_3}(s, c, 24, 25), \neg b(c, 74), \neg a(c, 25), a(s, 24), b(s, 75), f(s, c), c(99)\}$

$E_3 \supseteq \{\neg del_{\phi_1}, \neg del_{\phi_2}, del_{\phi_3}(s, c, 24, 25), a(c, 25), b(c, 74), a(s, 24), b(s, 75), f(s, c), c(99)\}$

According to Theorem 1, the corresponding logical specification with each extension is a minimal revised specification.

A temporary part in the specification corresponding with E_1:

$\phi_2 : b(c, 74)$.
$\phi_3 : f(F, E) \wedge a(F, A1) \wedge a(E, A2) \wedge A1 \leq A2 \supset \bot$.

A temporary part in the specification corresponding with E_2:

$\phi_1 : b(s, 75)$.
$\phi_3 : f(F, E) \wedge a(F, A1) \wedge a(E, A2) \wedge A1 \leq A2 \supset \bot$.

A temporary part in the specification corresponding with E_3:

$\phi_1 : b(s, 75)$.
$\phi_2 : b(c, 74)$.
$\phi'_3 : \neg(F = s, E = c, A1 = 24, A2 = 25) \wedge$
$f(F, E) \wedge a(F, A1) \wedge a(E, A2) \wedge A1 \leq A2 \supset \bot$.

4. PROOF PROCEDURE FOR COMPUTING MINIMAL REVISED SPECIFICATION

We can use a proof procedure proposed in [15][2]. Our procedure can check not only whether a formula is in an extension or not, but also accumulate justifications why a formula is in or out of an extension. Thus, we can compute a minimal revised specification by accumulating deletion literals in order to include a new specification without contradiction. However, the procedure proposed in [15] is a proof procedure for any arbitrary class of default logic, but in this paper, we only consider a normal default without prerequisites, so we can simplify the procedure. Figure 1 is such a simplified procedure. In the sequel, we explain this procedure.

Definition 6. Let α be a clause and Σ be a set of clauses. A *linear refutation proof* for α w.r.t. Σ is a sequence of clauses $C_0, ..., C_m$ such that:

1. $C_0 = \alpha$
2. $C_m = \Box$ (an empty clause)
3. C_i is a resolved clause of C_{i-1} and a clause in Σ or $C_0, ..., C_{i-1}$.

Definition 7. A *deletion assumption* Δ is a pair of sets of of deletion literals $\langle \Delta_{in}, \Delta_{out} \rangle$. We denote Δ_{in} and Δ_{out} as $in(\Delta)$ and $out(\Delta)$ respectively.

A deletion assumption is used to make a specification consistent. A literal in $in(\Delta)$ must be derived and a literal in $out(\Delta)$ must not be derived to keep consistency.

Definition 8. Let α be a clause. A linear refutation proof $C_0, ..., C_m$ for α is active w.r.t. a deletion assumption Δ if there is no $C_i \in out(\Delta)$.

[2]There are proof procedures for normal defaults of Default Logic [12, 11]. However, these methods only check whether a formula is in an extension or not and do not compute a justification to let the considered formula in/out of the extension.

If any deletion literal in $out(\Delta)$ is used during a linear refutation proof, the proof is not valid. Therefore, we need to check the activeness during the construction of the proof. This check actually prunes unnecessary proofs using deletion literals which were assumed to be out of an extension in a previous iteration.

In the procedure of Figure 1, we consider R_{new} as a formula in a temporary part and $DT_{CM}(\langle T_{pst}, T_{tmp} \cup \{R_{new}\} \rangle)$ as a intial default theory. Note that we can easily show that an extension without $del_{R_{new}}$ for the intial default theory exactly corresponds with an extension $DT_{CM}(\langle T_{pst} \cup \{R_{new}\}, T_{tmp} \rangle)$.

Let $\phi_{R_{new}}$ be a label of R_{new}.
We firstly call $out\_con(del_{\phi_{R_{new}}}, \langle \emptyset, \emptyset \rangle)$.

The initial call is to show that $del_{\phi_{R_{new}}}$ is out of an extension. To show that, firstly we add $del_{\phi_{R_{new}}}$ into $out(\Delta)$, consider every proof to derive $del_{\phi_{R_{new}}}$ and show that there exists the negation of a deletion literal for each proof such that the literal can be derived. This means that each proof does not become active in the extension. Then, to show that a deletion literal L is derived, we call $derive(L, \Delta)$. In $derive(L, \Delta)$, we find a proof for L such that for every negation of a deletion literal, $\neg del$, in the proof, del is out of an extension. Therefore, by calling out_con and $derive$ alternately, we calculate a set of deletion literals to let $del_{\phi_{R_{new}}}$ be out of the extension.

5. RELATED RESEARCH

In software engineering, there are several proposals of logical treatment of "inconsistency" of software specification [4, 3, 6]. A survey of these approaches is found in [9]. [4] handles the first systematic work on exception handling of integrity constraints in database specification and he proposes an isolation of such an exception from integrity constraints. [3] proposes a recovery of isolation when the exception is resolved for temporary violation of integrity constraints. Finkelstein et al. [6] use non-collapsible "quasi-classical logic" even in the existence of inconsistency and formalizes consistency management between multiple specifications defined by several users.

There are many researches on belief revision which would be related to consistency maintenance in software engineering. Fagin et al. [5] give a formalization of consistency management of a logical database consisting of first-order sentences.

Using default logic to express defeasible parts or uncertain parts of specification is not new. For example, Ryan [14] proposed a usage of order theory which is a prioritized default logic to represent entrenchment of specifications. Zowghi et al. [19] propose an application of default reasoning, belief revision and epistemic entrenchment to model requirements evolution. However, either of them do not give a computational method for consistency management.

However, in this paper, we not only formalize a minimal revision on specification, but aslo provide an implementation of minimal revised specifications in terms of default logic based on a new proof procedure. This procedure computes which hypotheses of deletion are necessary to achieve con-

```
out_con(d, Δ)
d: a deletion literal, Δ: a deletion assumption
begin
if d ∈ in(Δ) then fail
elseif d ∈ out(Δ) then return Δ
else
  Δ_0 = ⟨in(Δ), out(Δ) ∪ {d}⟩; i := 0;
  for every active linear refutation proof
      of W ∪ CONS(D) for ¬d w.r.t. Δ_i, C_0, ..., C_m do
  begin
    select negative deletion literal ¬d'
                                among C_j's (0 ≤ j ≤ m)
    if there is no such d' then fail
    if derive(d', Δ_i) returns Δ_{i+1} then
      i := i + 1; continue
  end
  return Δ_i
end /* out_con */

derive(d, Δ)
d: a deletion literal, Δ: a deletion assumption
begin
if d ∈ in(Δ) then return Δ
elseif d ∈ out(Δ) then fail
else
  select an active linear refutation proof
          of W ∪ CONS(D) for ¬d w.r.t. Δ, C_0, ..., C_m
  if there is no such proof then fail
  Δ_0 = Δ; i := 0;
  for every negative deletion literal ¬d'
                                in C_j's(0 ≤ j ≤ m) do
  begin
    if out_con(d', Δ_i) returns Δ_{i+1} then
      i := i + 1; continue
  end
  return ⟨in(Δ) ∪ {d}, out(Δ)⟩
end /* derive */
```

Figure 1: Proof Procedure for Calculating Minimal Deletion Literals

sistency. Moreover, in this proof procedure, relevant parts of the added specification will be checked for consistency so that we can avoid a computation of whole extensions which might have irrelevant parts of the added specification.

Inoue[7] proposes a similar technique to [16, 17] to maintain consistency. He translates a computation of his proposed extended abduction into a computation to ordinary abduction. However, to guarantee minimal updates, he needs a minimality check as [16, 17], so the problem of [16, 17] is inherited in his framework.

There is another technique of computing a minimal revised specification using minimal hitting sets[10, 1]. Firstly, they compute all sets of rules which lead to contradiction and then, they compute a minimal hitting set of all sets (a minimal set which has a common element with each set). However, this method has a problem of calculating all sources of contradiction in the first place instead of checking minimality in the last phase in [16, 17]. On the other hand, in our proof procedure, we can prune non-active proofs by deletion literals which are already assumed to be out of an extension, so that we do not need to check all sets of cotradictory rules.

In [2], they restrict a deltable rule to a fact in extentional data base and propose a method to avoid contradiction using tableau method. In this method, neither minimality check nor computing minimal hitting sets is necessary, but they do not consider a deletion of an arbitrary rule.

6. CONCLUSTION

The contributions of this research are the following.

- We propose a correct and complete method of computing a minimal revised specification by translating the specification into a default theory.
- We propose a simplified top-down proof prcedure to compute deletion literals to avoid consistency.

For a future work, we need to investigate an efficient implementation of the proof procedure and application to a real problem.

Acknowledgements We would like to thank anonymous referees for constructive comments on this paper. We also thank Aditya Ghose who gives us comments on the paper and related references.

7. REFERENCES

[1] Aravindan, C. and Baumgartner, P., A Rational and Efficient Algorithm for View Deletion in Databases., *Proc. of ILPS-97* pp. 165 – 179 (1997).

[2] Aravindan, C. and Baumgartner, P., Theorem Proving Techniques for View Deletion in Databases, *Journal of Symbolic Computation* Vol. 29, No. 2, pp. 119 – 147 (2000).

[3] Balzer, R., Tolerating Inconsistency, *Proc. of ICSE-13*, pp. 158 – 165 (1991).

[4] Borgida, A., Language Features for Flexible Handling of Exceptions in Information Systems, *ACM Transactions on Database Systems*, **10**, pp. 565 – 603 (1985).

[5] Fagin, R., Ullman, J. D., and Vardi, M. Y., On the Semantics of Updates in Databases, *Proc. of PODS'83*, (1983).

[6] Finkelstein, A. C. W., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B., Inconsistency Handling in Multiperspective Specifications, *IEEE Transactions on Software Engineering*, 20, pp. 569 – 578 (1994).

[7] Inoue, K., A Simple Characterization of Extended Abduction, *Proc. of CL2000* pp. 718 – 732 (2000).

[8] Lientz, B. P., and Swanson, E. B., Software Maintenance Management, Addison Wesley (1980).

[9] Nuseibeh, B., To Be and Not to Be: On Managing Inconsistency in Software Development, *Proc. of 8th IEEE International Workshop on Software Specification and Design (IWSSD-8)* pp. 164 – 169 (1996).

[10] Pereira, L.M., Damásio, C. V., Alferes, J. J., Diagnosis and Debugging as Contradiction Removal, *LPNMR-93*, pp. 334 – 348 (1993).

[11] Poole, D., Compiling a Default Reasoning System into Prolog, *New Generation Computing*, Vol. 9(1), pp. 3 – 38 (1991).

[12] Reiter, R., A Logic for Default Reasoning, *Artificial Intelligence* **13** pp. 81 – 132 (1980).

[13] Reiter, R., A Theory of Diagnosis from First Principles, *Artificial Intelligence* **32** pp. 57 – 95 (1987).

[14] Ryan, M., Defaults in specifications, Proc. of IEEE International Symposium on Requirements Engineering (RE'93), pp. 142 – 149 (1993).

[15] Satoh, K., A Top Down Proof Procedure for Default Logic by Using Abduction, *Proceedings of the Eleventh European Conference on Artificial Intelligence*, pp. 65 – 69, Amsterdam, the Netherlands (1994).

[16] Satoh, K., Computing Minimal Revised Logic Program by Abduction, *Proc. of the International Workshop on the Principles of Software Evolution*, pp. 177 – 182 (1998).

[17] Satoh, K., "Consistency Management in Software Engineering by Abduction", *Proceedings of the ICSE-2000 Workshop on Intelligent Software Engineering*, pp. 90 – 99, Limerick, Ireland (2000).

[18] Satoh, K., Computing Minimal Belief Revision by Extended Logic Programming without Minimality Check, *Proc. of IJCAI-01 Workshop on Abductive Reasoning*,pp. 48 – 55 (2001).

[19] Zowghi, D., Ghose, A., and Peppas, P., "A Framework for Reasoning about Requirements Evolution", *Proc. of PRICAI'96*, pp. 157 – 168 (1996).

Intelligent Support for Developing Adaptable Software Architectures: A Knowledge-Based Approach

Nary Subramanian
Dept. of Computer Engineering
Hofstra University
Hempstead, NY 11549
koolnary@yahoo.com

Lawrence Chung
Dept. of Computer Science
University of Texas at Dallas
Richardson, TX 75081
chung@utdallas.edu

ABSTRACT

Intelligent techniques can help practitioners generate adaptable software architectures, the first step in the development of adaptable software solutions. In this paper, we present the ASA (Adaptable Software Architecture) Framework, which is intended to help systematically generate adaptable software architectures from the corresponding adaptability requirements, treated as one type of non-functional requirements (NFRs). Taking the premise that at the core of intelligence lies the capability to represent and reason about knowledge of the subject matter, the ASA Framework offers techniques for representing knowledge of adaptability requirements and adaptability-enhancing architectural designs and for reasoning about them. The framework specializes the NFR Framework by focusing on adaptability requirements, uses Telos as the underlying knowledge representation language, and deploys ConceptBase as the knowledge base management system in providing a tool support - the ASA Assistant. The ASA Assistant uses case-based reasoning for retrieving architectural constituents from the knowledge base. The ASA Framework has been applied in three different systems, and an initial feedback has been obtained from domain experts.

Categories and Subject Descriptors

Software – Software Engineering – Requirements/Specifications (D.2.1); Software - Software Engineering - Software Architectures (D.2.11); Software - Software Engineering - Design Tools and Techniques (D.2.2).

General Terms

Design

Keywords

Adaptability, Software Architecture, Framework, Case-Based Reasoning, Knowledge Base, Tool, Non-Functional Requirements

1. INTRODUCTION

Intelligent techniques such as knowledge-based systems, reasoning systems, agent - based systems, etc. are expected to be useful for software engineering [5, 15, 25]. One of the interesting challenges in software engineering is systematically developing systems that satisfy non-functional requirements (or NFRs) such as adaptability, testability, reliability, and so on. Adaptability is emerging as an important NFR for software systems [6, 14, 18, 19]. Briefly stated, adaptability is the ability of software systems to accommodate changes in their environment. There is a wide belief that NFRs such as adaptability should be engineered into the software architecture itself, the first step in the development of software solutions, in order for the final software system to be adaptable. Intelligent techniques can help practitioners generate adaptable software architectures.

However, there are only a few techniques available to methodically develop adaptable software architectures. Attribute Tradeoff Analysis Method (ATAM) [10] is one such method. In ATAM, for each architecture developed, its tradeoffs against a previously determined list of attributes are determined and the most appropriate architecture(s) are chosen as a result of this exercise. Architecture Level Modifiability Analysis (ALMA) [12] is another technique for developing modifiable architectures. Each architecture is tested for its modifiability based on change scenarios and the architecture that best satisfies the pre-set goals is chosen. The Architecture Design Method (ADM) [2] is another method. In ADM adaptability is given a value; the value for adaptability for each architecture is estimated using different techniques including simulation, and the architecture that matches or exceeds the initial assigned value for adaptability is chosen. However, in all these techniques we find that there is a general lack of the ability to represent various software artifacts and to reason about them. In this paper we take the premise that at the core of intelligence lies the capability to represent and reason about knowledge[1] and the framework that we present, the Adaptable Software Architecture (ASA) Framework [22], offers techniques for representing knowledge of adaptability requirements and adaptability-enhancing architectural designs and for reasoning about them. The ASA Framework also helps to systematically generate adaptable software architectures from the corresponding adaptability requirements.

The ASA Framework is a specialization of the NFR Framework [3, 17] for the NFR adaptability and for the level of software architectures, uses Telos [16] as the underlying knowledge representation language, and deploys ConceptBase [8] as the

[1] In [15] on pages 100-102 it states that representation of knowledge is required by an intelligent system to support reasoning.

knowledge base management system (KBMS) in providing a tool support called the ASA Assistant. The ASA Assistant also uses case-based reasoning (CBR) for matching search criteria to the knowledge base content[2]. The ASA Framework was validated by applying it to generate adaptable software architectures for three different systems, implementing the architectures in the three systems, and by obtaining the initial feedback from domain experts. However, the results are preliminary and the Framework needs to be tested on a larger scale.

Our survey of the literature [23] suggests that there is no one definition for adaptability. In order to place the discussions in this paper in its proper perspective, we give our definition of adaptability that is comprehensive enough to accept most of the other definitions. Adaptation means change in the system to accommodate change in its environment. More specifically, adaptation of a software system (S) is caused by change (δ_E) from an old environment (E) to a new environment (E'), and results in a new system (S') that ideally meets the needs of its new environment (E'). Adaptability then refers to the ability of the system to make adaptation. Adaptation involves three tasks:

1. ability to recognize δ_E
2. ability to determine the change δ_S to be made to the system S according to δ_E
3. ability to effect the change in order to generate the new system S'.

Another concept that needs to be clarified is software architecture. Software architecture is the underlying structure of software systems. Software architecture has the following constituents [1, 20]: components, connections, patterns, constraints, styles, and rationales. Components are the elements from which systems are built; connections are the interactions between the elements; patterns describe the layout of the components and connections; constraints are on the components, connections and patterns; styles are an abstraction of architectural components from various architectures; and rationales describe why the particular architecture was chosen. Thus, for example, Interrupt Handler and Parser could be components, message passing could be the connection between them, style could be layered, constraint could be that interrupt handler should accept all interrupts and that the data received from the interrupt handler should be sent to the parser within 100ms (constraint on connection), pattern could be sequential processing, and rationale could be familiarity with this architecture.

Section 2 describes the ASA Framework, section 3 describes the use of case-based reasoning to develop adaptable architectures, section 4 discusses validation of the Framework, and section 5 presents the conclusions.

[2] Comparing our approach to the OMG MDA [9] effort, the latter develops a UML model of a system; the ASA Assistant can be used to develop this UML model of the software system itself – ASA Assistant helps in representing adaptability goals and reasoning about them, and also in searching for adaptable elements for developing the UML model.

2. THE ASA FRAMEWORK

The ASA Framework is a specialization of the NFR Framework. The NFR Framework has been influenced by work on decision support systems [4, 13], but the NFR Framework focuses on systematically dealing with non-functional requirements during system/software development. As a specialization of The NFR Framework, the ASA Framework further focuses on adaptability requirements and the various methods which can be used to satisfice such requirements.

The ASA Framework has the following components:

1. a means for representing and reasoning about adaptability requirements and software architectures,
2. a means for converting the representations into a KBMS format, and
3. a means for generating adaptable architectures using the KBMS.

Each of these components will be discussed in this section.

2.1 Representation and Reasoning Mechanisms

2.1.1 Softgoal Interdependency Graphs

In the ASA Framework, adaptability requirements are treated as softgoals[3] to be achieved during the process of software architecture development. Since the definition of adaptability itself is not fixed, for each project or domain the appropriate definition of adaptability is captured by the corresponding softgoal decomposition. This creates an NFR softgoal hierarchy that defines adaptability for the particular domain. The various software architectures and their constituents are represented by design (or operationalizing) softgoals. Satisficing of the NFR softgoals by the design softgoals can occur in different intensities, also called contributions of the design softgoals to the NFR softgoals, and are given in Figure 1. The reasons for the contributions are captured by the third type of softgoal – the claim SIG) and an example SIG is shown in Figure 2. The ontology used

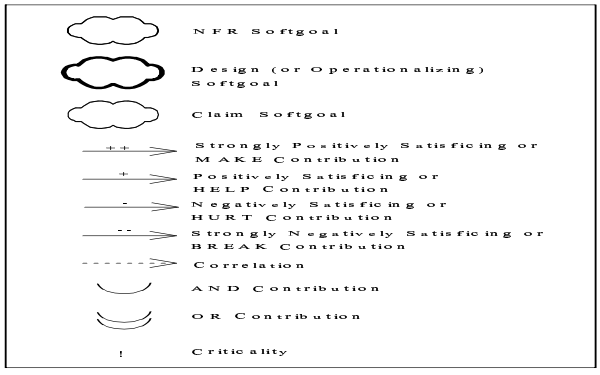

Figure 1. Ontology (Partial) of the ASA Framework

[3] i* [26] adopts the notion of softgoal to agent orientation but defines a softgoal as a condition in the world which the actor wishes to achieve, although the criteria for the condition being achieved is not sharply defined a priori but subject to interpretation.

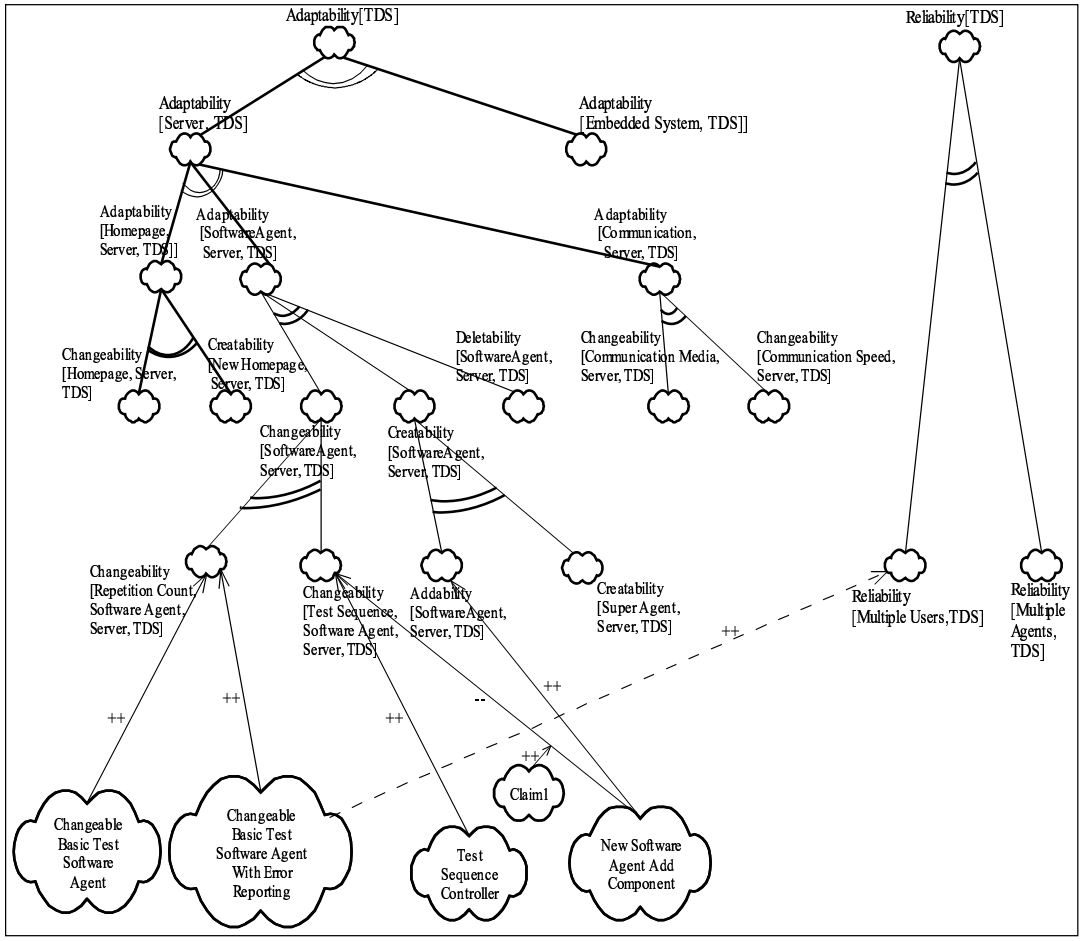

Figure 2. An Example SIG

softgoal. Sometimes a design softgoal may make synergistic or conflicting contributions to several NFR softgoals and these are captured by correlations. The graph created by softgoals and their contributions is called the Softgoal Interdependency Graph (or for the different SIG elements is given in Figure 1. In Figure 2, TDS stands for a type of system (Telepresent Diagnostic System) and the design softgoals represent some components in the domain of telepresent [21] diagnostics. Each softgoal is named using the convention

Type[Topic1, Topic2, ...]

where *Type* is an NFR and *Topic* is a system to which *Type* applies. Softgoals can also have criticalities associated with them – in some situations some softgoals may be more critical than the others.

Figure 2 is read as follows. Starting from the top, the NFR softgoal (indicated by the cloud) Adaptability[TDS], which represents the NFR adaptability as applicable to TDS, is OR-decomposed (indicated by the double arc) into Adaptability[Server, TDS] and Adaptability[Embedded System, TDS] child softgoals. A TDS system has both a server part and an embedded system that can be telepresently controlled over the internet by a client connected to the server. Adaptability of TDS can refer to the adaptability of either the server or the embedded system (or both) and hence the OR-decomposition. The sequence of decomposition continues till the leaf NFR softgoals are obtained. The design softgoals of interest are indicated at the bottom of Figure 2 (by the dark clouds) and they represent components developed in the TDS domain. The extent to which the various components satisfice the different NFR softgoals is captured by the contributions represented by the arrows from the design softgoals to the NFR softgoals. The strength of satisficing is indicated next to the arrows following the legend of Figure 1. The reasons for the strengths are captured by the claim softgoals (indicated by dashed cloud) in Figure 2. The cloud named "Claim1" could actually stand for "New Software Agent Add Component helps add new software agents and hence strongly satisfices changeability requirement". In Figure 2 we find another contribution indicated by the dashed arrow – the correlation: this captures the conflicting or synergistic interaction between NFRs; thus while the Changeable Basic Test Software Agent with Error Reporting strongly positively satisfices the NFR changeability it also strongly positively correlates with the NFR reliability which means that this component synergistically interacts with the two NFRs: changeability and reliability. Thus the requirements for the domain, the design elements and the rationale for the satisficing of the design elements are succinctly captured by the SIG.

2.1.2 Evaluation Process

The reasoning part of the ASA Framework is partly captured by the labeling and label propagation algorithms (the reasoning part is also performed using case-based reasoning that is discussed later). This process is interactive and can be performed by the developer. Briefly the procedure is as follows: the design softgoals are assigned labels 'S' for satisficed and 'D' for denied depending on whether the softgoal (which perhaps represents an architecture or its constituent) is satisficeable or deniable. These labels are decided by the developer based on available evidence – a satisficeable softgoal has enough justification to be acceptable for the design process, while for a deniable softgoal this is usually not true. Once the design softgoals have been assigned labels, these labels can be propagated to the NFR softgoals based on the contributions of the design softgoals to the NFR softgoals. The label propagation up the SIG usually takes place in two steps: the first step computes the individual impact of each satisficed interdependency on the parent, and the second step combines the individual impacts of all interdependencies at the parent into a single label for the parent. The architecture that produces the most satisfactory labels for the relevant adaptability softgoals is then chosen. The details of this process can be seen in [3]. Some example rules are:

1. if a softgoal is labeled 'S' and its contribution to its parent is MAKE or HELP then the parent softgoal's (partial) label is 'S'
2. if a parent softgoal gets the partial label of 'S' from all its child softgoals then the parent softgoal's final label is 'S'.

While we have not shown the labels in Figure 2, it can be assumed that all the design softgoals satisfice (this is usually decided by the developer - if the design softgoals positively satisfice then this is a safe assumption; if they negatively satisfice then the developer can make the decision) the leaf NFR softgoals. Because of the preceding propagation rules, all the design softgoals satisfice the root NFR softgoal Adaptability[TDS] as well. In other words, the design elements in Figure 2 are adaptable and the developer has the reasons for adaptability recorded in the claim softgoals. This also indicates one of the other advantages of the ASA Framework – its process orientation. We can determine the extent to which components satisfice the NFR softgoals during the process of architecture development. We do not need the completed architecture to evaluate its adaptability. If during this evaluation the developer finds a component to be unsatisfactory, the developer can remove that component from consideration and focus on another component that is perhaps more adaptable.

2.2 Converting to KBMS Format

Any SIG has several elements: softgoals, decomposition methods, operationalization methods, argumentation templates and correlation rules. Each of these SIG elements can be represented using frame-like notation. These frame-like notations can then be converted into a knowledge representation language such as Telos [16]. The advantages of Telos include treatment of link types as object, extensibility in the form of support for meta classes, and a strong theoretical foundation – temporal calculus and Church's type theory. Softgoals can be NFR softgoals, operationalizing softgoals or claim softgoals. Decomposition methods capture the relationships between child and parent softgoals where the child and parent softgoals are of the same type – either NFR softgoals or design softgoals. Operationalization methods capture the contributions of design softgoals to NFR softgoals; argumentation templates capture the claim softgoals; and correlation rules represent the correlations. Figure 3 depicts the frame-like notations for an NFR softgoal, an NFR decomposition method, an operationalization method and a correlation rule. These frame-like notations can be converted into the corresponding Telos descriptions in a straight forward manner. The Telos descriptions can then be directly populated in a KBMS. We used ConceptBase [8] for the KBMS and ConceptBase supports Telos notation. Elements of several SIGs for different domains were populated in the KBMS.

NFR Softgoal AdaptabilityTDS **Type: Adaptability** **Topic: TDS** **Priority: Low** **Label: Undecided** **Author: Nary** **Creation Time: May 20, 2003**	**NFRDecompositionMethod AdaptabilityTDSVia SubTopic** **Parent: Adaptability[TDS]** **Offspring:** **Adaptability[Server, TDS],** **Adaptability[Embedded System, TDS]** **Contribution: OR**
OperationalizationMethod TestSequenceController **Parent:** **Changeability [Test Sequence, Software Agent, Server, TDS]** **Offspring:** **TestSequenceController** **Contribution: MAKE** **ApplicabilityCondition: Always** **Constraint: Nil**	**CorrelationRule R1** **Parent: Reliability [Multiple Users, TDS]** **Offspring:** **ChangeableBasicTest SoftwareAgentWith ErrorReporting** **Contribution: MAKE** **ApplicabilityCondition: Always**

Figure 3. Frame-like Notations for some SIG Elements

2.3 Generating Adaptable Architectures Using KBMS

Once the KBMS has been populated with various SIG elements, for any new project the knowledge base (KB) can be searched for adaptable architectural constituents. The ASA Framework uses the idea that adaptable architectural constituents helps generate adaptable architectures. Therefore by first populating the KB with adaptable constituents for different domains, it will be possible to use the KB for developing adaptable architectures for new projects. This is another type of reuse but with the proviso that the reuse is guided by adaptability. Using the outputs of the KB, the architect can create an architecture. The adaptability of this architecture can again be tracked using SIGs for the (possibly new) application.

In order to put the constituents obtained from the KB together into an architecture we developed some heuristics based on our experience. Some of these are:

1. based on the specifications[4] of the constituents retrieved from the knowledge base in the architecture repository choose the constituents whose specifications suit the application being developed

[4] We used the DisCo (short for Distributed Cooperation) [7] specification language for recording specifications of architectural constituents.

2. components can be used as the starting point – develop the architecture by adding new components to complete the functionality not provided by the searched component(s)
3. using the other constituents (connections, patterns, constraints, styles and rationales) searched for complete the architecture(s).

3. CASE-BASED REASONING FOR GENERATING ARCHITECTURES

There are several techniques to search the KB. We have chosen to use case-based reasoning (CBR) [11] for this purpose. CBR is especially useful where knowledge is incomplete and adaptable architecture development seems to fit this requirement.

3.1 What is a case?

The search criteria that we were interested in were one or more of type, topic and contribution type of an architectural constituent. This constituted the problem space. The result of the search was the constituent and this represented the solution space. In order to use CBR a case-base was developed based on the contents of the KB. Each case in the case-base was of the form

(problem, solution)

where for each problem in the problem space the corresponding solution from the solution space was paired.

Once the KB is populated with the methods and correlation rules, a search through the KB can help create SIGs for various design softgoals in the KB. These SIGs were used as the basis for creating cases. Each case had only one design softgoal. This softgoal and its type (component, connection, pattern, constraint, style or rationale) formed the solution part of the case. The contribution of the design softgoal to its parent NFR softgoal, the types, and the topics of all its parent NFR softgoals in the SIG formed the problem space. Also part of the problem space were the correlation types that the design softgoal involved itself in, the types, and topics of the parent NFR softgoals of the correlations. Figure 4 illustrates case formation from a SIG. The tool ASA Assistant that we developed creates these cases automatically. The current population of the KB has more than 100 NFR softgoals, 50 design softgoals, and about 300 cases. The tool takes about 15 minutes to create the case-base from the architectural knowledge.

3.2 Uses Cases to Generate Adaptable Architectures

An example case is given below (can be obtained from the SIG of Figure 2):

NFR Softgoal Types: Changeability, Adaptability
NFR Softgoal Topics: TestSequence, SoftwareAgent, Server, TDS
NFR Satisficing Type: MAKE
Correlation Softgoal Types: Reliability
Correlation Softgoal Topics: MultipleUsers, TDS
Correlation Satisficing Type: MAKE
Constituent Name: TestSequenceController
Constituent Type: Component

Several such cases exist in the KB. The search criteria can be one or more of NFR softgoal type, NFR softgoal topic, and NFR satisficing type. Optionally, correlation NFR softgoal type, correlation NFR softgoal topic, and correlation NFR satisficing type may also be given. Thus if the input criteria for architectural component were "changeability" for NFR softgoal type, "testsequence" for NFR softgoal topic and "MAKE" for NFR satisficing type, then the result of the search would be "TestSequenceController". Using such outputs and the heuristics of Section 2.3, adaptable architectures can be generated.

4. VALIDATING THE ASA FRAMEWORK

This section discusses validation of the ASA Framework. The ASA Framework was validated using the following techniques:

1. Developing a tool called the ASA Assistant.
2. Applying ASA Framework to develop adaptable architectures for three systems.
3. Obtaining feedback from domain experts.

4.1 ASA Assistant

We developed a tool called the ASA Assistant that helps to populate the KB and retrieve data from the KB. The ASA Assistant is based on the client-server style, runs on Unix, and has been written in Tcl/Tk. The KBMS used is ConceptBase that uses the Telos knowledge representation language. The ASA Assistant provides graphical user interfaces (GUIs) for populating the KBMS with the various architectural constituents, for displaying the contents of the KB and for searching the KB for different constituents. The ASA Assistant uses CBR for searching the KB and provides the user interface for updating the case-base. After populating the KB with adaptable architectural constituents for various domains, the user can choose to update the case-base, whereupon the ASA Assistant automatically creates the case-base from all the constituents in the KB. Once the case-base has been updated, the ASA Assistant is ready for use in a new project. Using the GUIs provided for searching the KB, the user can search the KB for appropriate constituents. The ASA Assistant uses icons for displaying the different constituents and for each output also indicates the extent of the match (in percentage).

4.2 Application of ASA Framework to Three Systems

The ASA Framework was applied to three systems for which adaptability was an important NFR. For each system different adaptation problems were chosen and adaptable architectures were generated for each problem using the ASA Assistant. The architectures generated were implemented in the actual system and confirmed to be adaptable. This confirmation was again kept track of by means of a SIG. The three systems were vocabulary evolution system (VES), the user interface system (UIS) and the adaptable remote maintenance system (ARMS). The VES [24] supports evolution of syntax used to communicate system; the UIS provides means of entering data and displaying outputs; and the ARMS helps in remotely maintaining (and controlling) an embedded system.

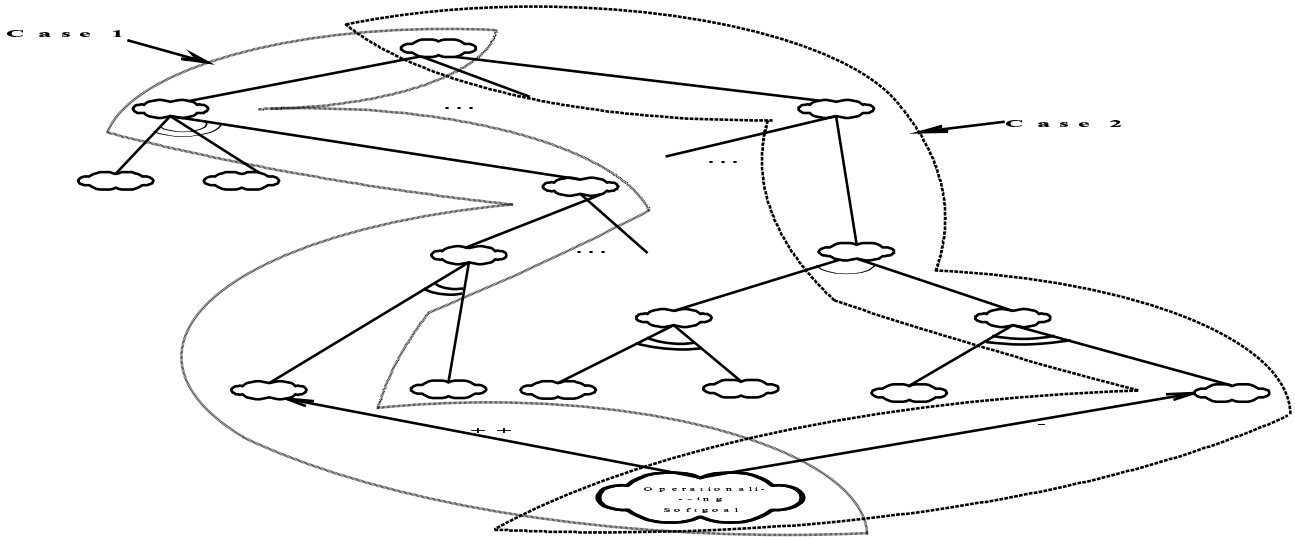

Figure 4. Deriving Cases from a SIG

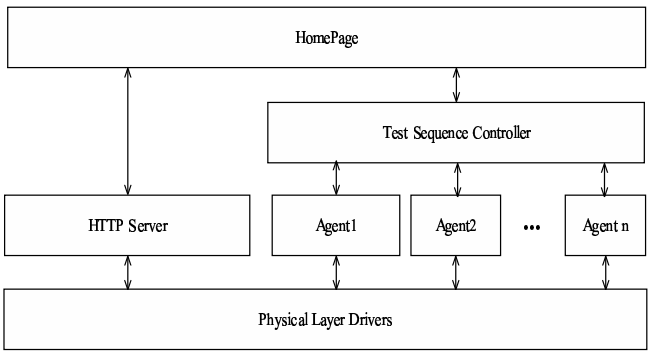

Figure 5. Architecture Generated Using the ASA Framework

Architecture[5] generated for TDS (a variant of ARMS) is given in Figure 5 which uses a component *Test Sequence Controller* obtained from the ASA Assistant.

4.3 Feedback from the Domain Experts

We obtained feedback on ASA Framework from several experts in the industry with a cumulative experience of over 50 years in software development. The feedback was obtained by means of questionnaires, presentations, email, telephone conversations, and face-to-face discussions. The general positive viewpoints were: adaptability is important enough to warrant the ASA Framework; the framework promotes component reuse; the implicit process instituted by the framework is practical; the fact that the framework accepts almost any definition of adaptability is good; and the framework will be useful for architects, system engineers, developers and process engineers. On the negative side the experts felt that training for potential users needs to be provided; motivation needs to be provided to change current practices; the effort needed to develop and maintain the KB may be excessive; process maturity may be needed in an organization to use the framework; and that the framework needs to be tested on a larger scale. We believe that the feedback provides a valuable "reality check" on the practicality of ASA Framework.

5. CONCLUSION

Adaptability is becoming an important non-functional requirement (NFR) for software systems. In order to develop an adaptable software system its software architecture should itself be adaptable in the first place. However, there are very few systematic methods to develop adaptable software architectures. In this paper we have presented the Adaptable Software Architecture (ASA) Framework [22] that helps to generate adaptable software architectures using intelligent techniques: provides capability for representing and reasoning about software artifacts required to generate adaptable architectures, provides ability to convert the representations into frame-like notations that can be captured by a knowledge representation language called the Telos [16] in a straight-forward manner, allows for storing the Telos representations in a KBMS called ConceptBase [8], and uses case-based reasoning for searching the stored adaptable architectural constituents (components, connections, patterns, constraints, styles and rationales) from the KBMS. The ASA Framework was validated by implementing a supporting tool called the ASA Assistant, using the Assistant in developing adaptable architectures for three systems, and obtaining feedback from domain experts.

There is still much work left to be done – the drawbacks indicated by the domain experts needs to be put right, the ASA Assistant needs modification for dynamic case-base updating (upon changes to the KB), applying the ASA Framework in more systems, providing tool support for Softgoal Interdependency Graph (SIG) development, and architectural verification at the specification level (using DisCo Animator [7], for example). Other interesting

[5] Representing architectures itself is another area of research with different techniques such as box-and-line diagrams, modeling languages, and architecture description languages being used; we have used simple box-and-line diagram in Figure 5, where boxes represent components and arrows represent procedure calls in the direction of the arrow.

avenues for future work include considering application of the ASA Framework to product-line architectures, and adapting the Framework to consider other NFRs such as security and safety. However, we feel that the ASA Framework promises to be an intelligent way of generating adaptable software architectures.

6. ACKNOWLEDGEMENTS

We thank the anonymous reviewers of the original version of this paper for their insightful comments.

7. REFERENCES

[1] Bass, L., Clements, P., and Kazman, R. Software Architecture in Practice. SEI Series in Software Engineering, Addison-Wesley, 1998.

[2] Bosch, J., and Molin, P. Software Architecture Design: Evaluation and Transformation. Proceedings of IEEE Conference and Workshop on Engineering of Computer-Based Systems, March 1999, pp. 4-10.

[3] Chung, L, Nixon, B. A., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Boston, 2000.

[4] Conklin, J., and Begeman, M. L. gIBIS: A Hypertext Tool for Explanatory Policy Decisions. ACM Transactions on Office Information Systems, Vol. 6, Issue 4, pp. 303- 331, 1988.

[5] Damiani, E., Khosla, R., and Kitjongthawonkul, S. A Human-Centered Approach for Intelligent Internet Applications. Soft Computing Agents: New Trends for Designing Autonomous Systems, (Eds.) V. Loia and S. Sessa, Physica-Verlag, Heidelberg, 2001, pp. 169 – 190.

[6] Hayes-Roth, B. Making Intelligent Systems Adaptive. Architectures for Intelligence, (Ed.) K. VanLehn, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1991,301 – 322.

[7] http://disco.cs.tut.fi

[8] http://www-i5.informatik.rwth-achen.de/CBdoc/cbaccess.html

[9] http://www.omg.org/mda

[10] Kazman, R., Klein, M., and Clements, P. ATAM: Method for Architecture Evaluation. CMU-SEI Technical Report CMU/SEI-2000-TR-004, 2000.

[11] Kolodner, J. Case-Based Reasoning. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[12] Lassing, N., Bengtsson, P., Hans van Vliet and Bosch, J. Experiences with ALMA: Architecture-Level Modifiability Analysis. The Journal of Systems and Software, Vol. 61, 2002, pp. 47-57.

[13] Lee, J. A Decision Rationale Management System: Capturing, Reusing, and Managing the Reasons for Decisions. Ph.D. Thesis, MIT, 1992.

[14] Loia, V., and Sessa, S. A Soft Computing Framework for Adaptive Agents. Soft Computing Agents: New Trends for Designing Autonomous Systems, (Eds.) V. Loia and S. Sessa, Physica-Verlag, Heidelberg, 2001, pp. 191 – 220.

[15] Meystel, A. M., and Albus, J. S. Intelligent Systems: Architecture, Design, and Control. John Wiley & Sons, New York, 2002.

[16] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. Telos: Representing Knowledge About Information Systems. ACM Transactions on Information Systems, Vol. 8, No. 4, October 1990, pp. 325-362.

[17] Mylopoulos, J., Chung, L., Liao, S. S. Y., Wang, H., and Yu, E. Exploring Alternatives During Requirements Analysis. IEEE Software, January/February 2001, pp. 1- 6.

[18] Neidhoefer, J. C., and Krishnakumar, K. Intelligent Control for Near-Autonomous Aircraft Missions. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, Vol. 31, No. 1, January 2001, pp. 14-29.

[19] Perlman, G. Achieving Universal Usability by Designing for Change. Internet Computing, Vol. 6, No. 2, March/April 2002, pp. 46-55.

[20] Shaw, M., and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.

[21] Steur, J. Defining Virtual Reality: Dimensions Determining Telepresence. Journal of Communication, Vol.42, No.4, 1992, pp. 73-93.

[22] Subramanian, N. Adaptable Software Architecture Generation Using The NFR Approach. Ph.D. Thesis, University of Texas at Dallas, Richardson, Texas, 2003.

[23] Subramanian, N., and Chung, L. Software Architecture Adaptability – An NFR Approach. Proceedings of Fourth International Workshop on Principles of Software Evolution (IWPSE2001), Vienna, September 2001, ACM Press, pp. 52-61.

[24] Subramanian, N., and Chung, L. Architecture-Driven Embedded Systems Adaptation for Supporting Vocabulary Evolution. Proceedings of the International Symposium on Principles of Software Evolution, IEEE Computer Press, Nov. 2000, pp. 144 – 153.

[25] Venkatram, P. Elements of Computational Intelligence for Network Management. Computational Intelligence in Telecommunication Networks, (Eds.) W. Pedrcyz and A. Vasilokos, CRC Press, Boca Raton, Florida, 2001, pp. 329 – 361.

[26] Yu, E. Modelling Strategic Relationships for Process Reengineering. Ph.D. Thesis, University of Toronto, 1995.

Reasoning about Requirements Evolution using Clustered Belief Revision

O. Rodrigues[ω], A. d'Avila Garcez[δ], A. Russo[ρ]
[ω]Department of Computer Science, King's College London, UK, odinaldo@dcs.kcl.ac.uk
[δ]Department of Computing, City University London, UK, aag@soi.city.ac.uk
[ρ]Department of Computing, Imperial College London, UK, ar3@doc.ic.ac.uk

ABSTRACT

During the development of system requirements, software system specifications are often inconsistent. Inconsistencies may arise for different reasons, for example, when multiple conflicting viewpoints are embodied in the specification, or when the specification itself is at a transient stage of evolution. These inconsistencies cannot always be resolved immediately. As a result, we argue that a formal framework for the analysis of evolving specifications should be able to *tolerate* inconsistency by allowing reasoning in the presence of inconsistency without trivialisation, and *circumvent* inconsistency by enabling impact analyses of potential changes to be carried out. This paper shows how *clustered belief revision* can help in this process. Clustered belief revision allows for the grouping of requirements with similar functionality into clusters and the assignment of priorities between them. By analysing the result of a cluster, an engineer can either choose to rectify problems in the specification or to postpone the changes until more information becomes available.

1. INTRODUCTION

Conflicting viewpoints inevitably arise in the process of requirements elicitation. Conflict resolution, however, may not necessarily happen until later in the development process. This highlights the need for requirements engineering tools that support the management of inconsistencies [15, 19].

Many formal methods of analysis and elicitation rely on Classical Logic as the underlying formalism. Model Checking, for example, typically uses temporal operators on top of classical logic reasoning [11]. This facilitates the use of well-behaved and established theorems and proof procedures. On the other hand, Classical Logic does not accept inconsistency, in the sense that one can derive anything from an inconsistent theory. For example, one can derive any proposition B from propositions A and $\neg A$. This is known as *theory trivialisation*, and is clearly undesirable in the context of requirements engineering, where inconsistency often arises [6, 10].

Paraconsistent Logics [3] attempt to ameliorate the problem of theory trivialisation by weakening some of the axioms of classical logic, often at the expense of reasoning power. For instance, Belnap's four valued logic [2] allows for non trivial logical representations where propositions can be both true and false, but does not verify basic inference rules such as Modus Ponens. While appropriate for concise modelling, logics of this kind are too weak to support practical reasoning and the analysis of inconsistent specifications.

Clustered belief revision [17] takes a different view and uses theory prioritisation to obtain plausible (i.e. not trivial) conclusions from an inconsistent theory, yet exploiting the full power of classical logic reasoning. This allows the requirements engineer to analyse the results of different possible prioritisations by reasoning classically, and to evolve specifications that contain conflicting viewpoints in a principled way. The analysis of user-driven cluster prioritisations can also give stakeholders a better understanding of the impact of certain changes in the specification.

In this paper, we investigate how clustered belief revision can support requirements elicitation and evolution. In particular, we have developed a tool for clustered revision that translates requirements given in the form of "if then else" rules into the (more efficient) disjunctive normal form (DNF) for classical logic reasoning and cluster prioritisation. We have then used a simplified version of the light control case study [9] to provide a sample validation of the clustered revision framework in requirements engineering.

The rest of the paper is organised as follows. In Section 2, we present the clustered revision framework. In Section 3, we apply the framework to the simplified light control case study and discuss the results. In Section 4, we discuss related work and, in Section 5, we conclude and discuss directions for future work.

2. CLUSTERED BELIEF REVISION

The AGM theory of belief revision is a formalism used to model the kind of information change in which an agent reasoning about his beliefs about the world is forced to adjust them in face of new (possibly contradictory) information. Work in the area started in the early 80's [1]. One of the main references to the work is the book "Knowledge in Flux" [8].

Clustered belief revision [17] is based on the main principles of belief revision but has two important features not present in the original work: the use of extra-logical information to help in the process of conflict resolution and the ability to group sentences with similar *role* into a *cluster*.

Clustered belief revision also uses sentences in DNF in order to make the deduction and resolution mech-

anisms more efficient. A cluster can be resolved and simplified into a single sentence in DNF. The resolution extends classical deduction by using the extra-logical information to decide how to solve the conflicts. Clusters can be embedded in other clusters and priorities between clusters can be specified in the same way that priorities can be specified within a single cluster. The embedding allows for the representation of complex structures which can be useful in the specification of requirements in software engineering.

The behaviour of the selection procedure in the deduction mechanism – that makes the choices in the resolution of conflicts – can be tailored according to the ordering of individual clusters and its intended local interpretation. We provide one such ordering based on the confidence/priority that the user has/wants to assign to each cluster.

Our approach has the following main characteristics: *i)* it allows users to specify groups of sentences associated with some possibly incomplete priority information. *ii)* it resolves conflicts in a group by taking into account the priorities specified by the user priorities and provides a consistent conclusion whenever possible *iii)* it allows groups to be embedded in other groups so that complex priority structures can be specified *iv)* and finally, combines the reasoning about the priorities with the deduction mechanism itself in an intuitive way.

In the resolution of a cluster, the main idea is to specify a deduction mechanism that reasons with the priorities and computes a *conclusion* based on these priorities. The priorities themselves are used only when conflicts arise, in which case sentences associated with higher priorities override those associated with lower priorities. The *prioritisation principle* used here is that "a sentence with priority x cannot block the acceptance of another sentence with priority higher than x". In the original AGM theory of belief revision, the prioritisation principle is applied only to the sentence the agent wants to incorporate in his beliefs: it is given the highest priority amongst all of the agents' belief.

The other principle used is that of *minimal change*. In the original AGM theory this amounts to require that the agent should not give up any of his old beliefs unless it is strictly necessary in order to repair the inconsistency caused by the new belief. In our case, we extend this a bit further since we have several levels of priority. We state it as follows: "information should not be lost unless it causes inconsistency with information conveyed by sentences with higher priority". As a result, when a cluster is provided with no relative priority between sentences, the mechanism computes a sentence whose models are logically equivalent to the models of the (union of) the maximal consistent subsets of the cluster. On the other extreme, if sentences in the base are linearly prioritised, the mechanism behaves in a way similar to Nebel's *linear prioritised belief bases*[1] [14].

DEFINITION 1. *A* labelled belief base *(LBB) is a tuple* $\mathsf{B} = \langle \mathcal{J}, \leq, f \rangle$*, where* $\mathcal{J}$ *is a set of labels,* $\leq$ *is a (partial) pre-order on* $\mathcal{J}$ *and* f *assigns elements of* $\mathcal{J}$ *to sentences of the language.*

DEFINITION 2. *A* structured cluster *is a tuple* $\Xi = \langle \mathcal{C}, \sqsubseteq, g \rangle$ *where* $\mathcal{C}$ *is a set of labels,* $\sqsubseteq$ *is a (partial) pre-order on* $\mathcal{C}$ *and* g *is a function assigning elements of* $\mathcal{C}$ *to either a sentence; a LBB or another cluster.*

[1]In our case, a belief base is a partial specification.

DEFINITION 3. *The level of a propositional logic formula is* 0*. Let* $\Xi = \langle \mathcal{C}, \sqsubseteq, g \rangle$ *be a cluster. The level of* Ξ*, in symbols* level(Ξ) *is defined recursively as* level(Ξ) $= max_{i \in \mathcal{C}}\{\text{level}(g(i))\} + 1$.

Thus, a cluster of level 1 is just a labelled belief base as defined previously.

DEFINITION 4. *Let* $K = \{\varphi_1, \ldots, \varphi_k\}$ *be a finite set of sentences. A* matrix representation of K *is obtained by associating rows of the matrix with logically equivalent formulas in DNF of each sentence where the columns are the disjuncts in those sentences.* $\mathcal{M}_K$ *will denote a chosen matrix representation of* K*.*

DEFINITION 5. *Let* $\mathcal{M}_K$ *be a matrix representation of a set* K*. A* path in $\mathcal{M}_K$ *is a set* $\wp$ *of disjuncts, each and only one taken from each row in* $\mathcal{M}_K$*. We denote the set of all paths in* $\mathcal{M}_K$ *by* paths($\mathcal{M}_K$)*.*

Note that a given path contains exactly one *representative* disjunct of each sentence in the belief base. Also, the order in which the sentences are laid out in the matrix is irrelevant, and so is the distribution of disjuncts of a given sentence in a row of the matrix.

DEFINITION 6. *The conjunction of all disjuncts visited in a path* $\wp$ *will be denoted by* $\sigma(\wp)$*. If* $\wp$ *is empty, we define* $\sigma(\wp)$ *to be* $\top$ *(we assume the language has a symbol for truth).*[2]

Ultimately what we want is to compare the best combinations of sentences in a belief base verifying prioritisation and consistency. If we can keep them all consistently so much the better, but this is not always possible. In order to compare subsets of the belief base, we define an ordering $\preceq$ on $2^{\mathcal{J}}$. We use $X \preceq Y$ to denote that the satisfiability X's sentences is at least as plausible as Y's. $X \prec Y$ means $X \preceq Y$ and $Y \not\preceq X$.

DEFINITION 7. *Let* $\mathsf{B} = \langle \mathcal{J}, \leq, f \rangle$ *be a labelled belief base and* $X, Y \in 2^{\mathcal{J}}$*.* $X \preceq Y$ *iff either* i) $Y = \emptyset$*; or* ii) $\exists x \in X$, $\exists y \in Y$, *s.t.* $x \leq y$ *and* $X - \{x\} \preceq Y - \{y\}$*; or* iii) $\exists x \in X$, $\exists Y' \subseteq Y$, *s.t.* $Y' \neq \emptyset$ *and* $\forall y \in Y'. x < y$ *and* $X - \{x\} \preceq Y - Y'$*.*

A number of results about the behaviour of $\preceq$, including compuational properties is given in [17]. We cannot list them all here for space reasons, but we include some in order to show that some of our expectations about $\preceq$ are met.

PROPOSITION 8. $\preceq$ *is a pre-order.*

PROOF. This proof is rather long and omitted here. See [17] for details. □

Notice that if one sentence in the base is the maximum w.r.t. to $\leq$, it is easy to show that condition iii) in Definition 7 will ensure that any subset of the base containing that sentence will be strictly preferred w.r.t. $\preceq$ to any subset *not* containing it. Therefore, if a new sentence (belief) is given the highest priority, our formalism verifies the basic principle of the primacy of the update of the original AGM theory of belief revision. This principle states that the new belief is always accepted in the result of a revision operation.

Definition 7 subsumes the converse of the the set-inclusion ordering, and hence accepting as many sentences whenever possible is guaranteed:

[2]Notice that $\sigma(\wp)$ is simply a conjunction of literals and the basis for rebuilding formulas in DNF. In model theoretical terms, $\sigma(\wp)$ represent one possible way of satisfying a belief base.

PROPOSITION 9. *$Y \subseteq X$ implies $X \preceq Y$.*

PROOF. Proof by induction on $|Y|$. If $|Y| = 0$, then $Y = \emptyset$ and then the proposition holds trivially. Assume that it holds for $|Y| = k$ and suppose $|Y| = k+1$. Take $x \in Y$. Since $Y \subseteq X$, $x \in X$. Therefore, $\exists x' \in X$ and $\exists y' \in Y$ such that $x' \leq y'$. For this we just set $x' = y' = x$. $|Y - \{y'\}| = k$ and $Y - \{y'\} \subseteq X - \{x'\}$, and hence, by the induction hypothesis, $X - \{x'\} \preceq Y - \{y'\}$. By Definition 7, $X \preceq Y$. □

However, the converse is not true. It might be the case that $X \preceq Y$, but $Y \subseteq X$ does not hold. It will depend on how important the elements in both sets are. A consequence of Proposition 9 is the following.

PROPOSITION 10. *Let $\langle \mathcal{J}, \leq \rangle$ be a partial pre-order. $\mathcal{J}$ is the minimum on $\langle 2^{\mathcal{J}}, \preceq \rangle$ and $\emptyset$, the maximum.*

PROOF. It is easy to see that $\mathcal{J}$ is the minimum: for any $X \in 2^{\mathcal{J}}.X \subseteq \mathcal{J}$. Therefore, by Proposition 9, $\mathcal{J} \preceq X$, for all $X \in 2^{\mathcal{J}}$. Similarly, $X \preceq \mathcal{J}$ only if $X = \mathcal{J}$. That $\emptyset$ is the maximum follows directly from Definition 7. □

Figure 1 gives examples of some orderings $\leq$ on $\mathcal{J}$ and the derived orderings $\preceq$ on $2^{\mathcal{J}}$. A connecting arrow from a to b indicates that $a < b$ or $a \prec b$ (i.e. that a is preferred to b).

Given a path in a matrix, we are interested in combinations of disjuncts in the path that are not contradictory (we will be especially interested in *maximal such combinations*):

DEFINITION 11. *Let $\mathcal{M}_K$ be a matrix representation of a set K and* paths$(\mathcal{M}_K)$ *the set of all paths in $\mathcal{M}_K$. The set of* consistent subpaths of $\mathcal{M}_K$ *is defined as* $\text{paths}^{\top}(\mathcal{M}_K) = \{\xi \mid \exists \wp \in \text{paths}(\mathcal{M}_K)$ *s.t.* $\xi \subseteq \wp$ *and* $\sigma(\xi)$ *is not contradictory*$\}$.

DEFINITION 12. *The* label set of a path $\wp$ *is the set* $\text{ls}(\wp) = \{\alpha \mid \alpha : P \in \wp\}$. *The* label abstraction of a set of paths Λ *is the set* $\text{La}(\Lambda) = \{\text{ls}(\wp) \mid \wp \in \Lambda\}$.

DEFINITION 13. *The* maximal plausible subpaths of a matrix representation of a set K *are the elements of the set* $\text{mps}(\mathcal{M}_K) = \{\wp \in \text{paths}^{\top}(\mathcal{M}_K) \mid \text{ls}(\wp)$ *is $\preceq$-minimal in* $\text{La}(\text{paths}^{\top}(\mathcal{M}_K))\}$.

DEFINITION 14. *Let $\mathsf{B} = \langle \mathcal{J}, \leq, f \rangle$ be a labelled belief base and $\mathcal{M}_{\mathsf{B}}$ a matrix representation of the sentences mapped by f in B. The result of flattening out B, in symbols* FLATTEN_BASE(B), *is the sentence* $\bigvee_{\xi \in \text{mps}(\mathcal{M}_{\mathsf{B}})} \sigma(\xi)$.

It is also possible to flatten out a cluster of higher order by recursively flatenning out all embedded subclusters as follows (this is simply an extension to Definition 14).

DEFINITION 15. *Let $\Xi = \langle \mathcal{C}, \sqsubseteq, g \rangle$ be a structured cluster. The result of flattening out Ξ, in symbols* FLATTEN_CLUSTER(Ξ), *is the sentence in DNF obtained in the following way:*

$$\text{FLATTEN\_CLUSTER}(\Xi) = \begin{cases} \text{FLATTEN\_BASE}(\Xi) & \text{if } \text{level}(\Xi) = 1 \\ \text{FLATTEN\_CLUSTER}(\Xi') & \text{otherwise} \end{cases}$$

where Ξ' is the cluster obtained from Ξ by replacing the function g by the function g', such that $g'(i) =$ FLATTEN_CLUSTER$(g(i))$, *for all $i \in \mathcal{C}$.*

EXAMPLE 1. *Consider the ordering $\leq$ in the middle of Figure 1 and assume that $f(x) = user\_in \wedge gte\_lux_2$, $f(w) = \neg user\_in \vee default\_lights'$, $f(y) = \neg default\_lights' \vee \neg no\_lights'$ and $f(z) = \neg gte\_lux_2 \vee no\_lights'$. These sentences correspond to the DNF of the sentences in the second inconsistency example discussed in the next section and the ordering $\leq$ is actually the same in prioritisation P1 there.*

The sentences taken conjunctively are inconsistent, so we would have to look for consistent subpaths in the matrix of this set. It can be shown that the consistent subpaths with highest number of elements will be associated with the labels in the sets $\{x, w, y\}$, $\{x, y, z\}$, $\{x, w, z\}$ and $\{w, y, z\}$. According to the ordering $\preceq$, the most plausible ones amongst those are $\{x, w, y\}$ and $\{x, y, z\}$. $\{w, y, z\}$ cannot be chosen as it does not contain a label of the most important sentence, namely x. $\{x, w, z\}$ is not chosen because it is strictly worse than $\{x, w, y\}$, since the latter contains y which is strictly better than z.

As a result, this ordering would produce a result which accepts sentences associated with x and y and includes the consequences of the disjunction of sentences w and z. This signals that whereas it is possible to consistently accept x and y, it is not possible to consistenly include both w and z. Given the assigned priorities, a choice between them cannot be made and their disjunction is taken instead.

3. THE LIGHT CONTROL EXAMPLE

In what follows, we adapt and simplify the Light Control Case Study [16] in order to illustrate the relevant aspects of our revision approach. The Light Control System (LCS) describes the behaviour of light settings in an office building. We consider two possible light scenes: a *default* light scene and a *chosen* light scene. Office lights are set to a default level upon entry of a user, who can then override this setting to a chosen light scene. If an office is left unoccupied for more than T_1 minutes, the system turns the lights off. When an unoccupied office is reoccupied within T_2 minutes, the light scene is re-established according to its immediately previous setting. The value of T_1 is set by the facilities' manager whereas the value of T_2 is set by the office user [9].

A dictionary of the symbols used in the LCS case study is given in Table 1. As usual, unprimed literals denote properties of the current state of the system, and primed literals denote properties of the next state (e.g., *occupied* denotes that a user is in the office at time t, and *occupied'* denotes that a user is in the office at time $t+1$).

A partial specification of the LCS is given below:

Behaviour rules

$r_1 : user\_in \rightarrow occupied'$
$r_2 : occupied \wedge user\_out \wedge \neg elapsed\_T_2 \rightarrow temp\_unocc'$
$r_3 : temp\_unocc \wedge elapsed\_T_1 \rightarrow unoccupied'$
$r_4 : temp\_unocc \wedge user\_in \rightarrow occupied'$
$r_5 : unoccupied \rightarrow no\_lights'$
$r_6 : temp\_unocc \wedge user\_in \rightarrow chosen\_lights'$
$r_7 : user\_in \rightarrow default\_lights'$

Rules r_5 to r_7 specify the intended behaviour of the office lights: *no_lights* indicates that the office lights are off; *chosen_lights* indicates that the office lights are as set by the user; and *default_lights* indicates that the office lights are in the default setting. We assume that the initial chosen light scene is set to the default one.

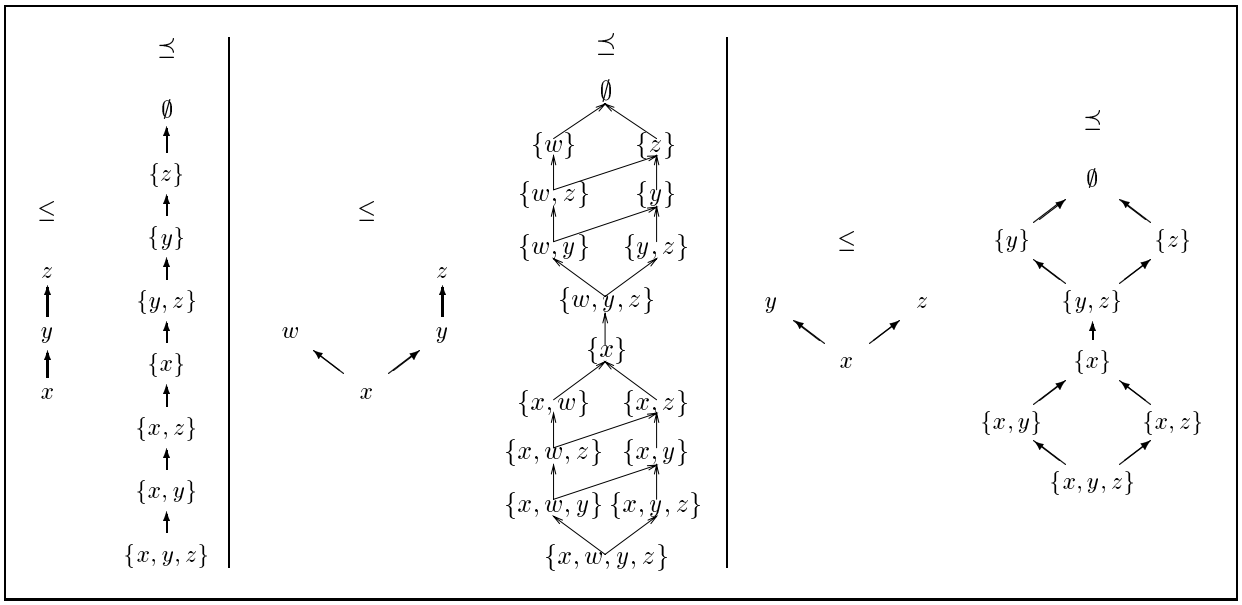

Figure 1: Examples of orderings $\leq$ on the clusters and the corresponding final ordering $\preceq$.

proposition	meaning
occupied	a user is in the office
user_in	a user enters an unoccupied office
user_out	a user leaves an office unoccupied
temp_unocc	the office is unoccupied for less than T_2 minutes
unoccupied	the office is unoccupied for T_1 minutes or more
$elapsed\_T_i$	T_i minutes have elapsed
chosen_lights	office lights are as set by the user
default_lights	office lights are in the default setting
alarm	the alarm is activated
gte_lux_1	day light level is greater or equal to the light level required by the chosen or default light scene (lux_1)
gte_lux_2	day light level is greater or equal to the maximum luminosity achievable by the office lights (lux_2)
no_lights	office lights are off

Table 1: Dictionary of symbols used in the specification.

In our study, we consider that the light control system should satisfy two types of properties: *safety* properties and *economy* properties. The following are safety properties: *i)* the lights are not off in the default light scene; *ii)* if the fire alarm ($alarm$) is triggered, the default light scene must be established in all offices; and *iii)* T_3 minutes after the alarm is triggered, the lights must all be turned off (i.e., only emergency lights must be on). The value of T_3 is set by the facilities manager. The above requirements are represented by rules s_1 to s_4:

Safety rules

$s_1 : alarm \wedge \neg elapsed\_T_3 \rightarrow default\_lights'$
$s_2 : alarm \wedge elapsed\_T_3 \rightarrow no\_lights'$
$s_3 : default\_lights \leftrightarrow \neg no\_lights$
$s_4 : default\_lights' \leftrightarrow \neg no\_lights'$

Economy properties include the fact that, to the extent feasible, the system ought to use natural light to achieve the light levels required by the office light scenes. Sensors can check *i)* whether the luminosity coming from outside is enough to surpass the luminosity required by the current light scene; and *ii)* whether the luminosity coming from outside is greater than the maximum luminosity achievable by the office lights. The latter is useful because it can be applied independently of the current light scene in an office. Let lux_1 denote the luminosity required by the current light scene, and lux_2 the maximum luminosity achievable by the office lights. The above can be summarised as follows: *i)* if the natural light is at least lux_1 (gte_lux_1) and the office is in the chosen or default light scene, then the lights must be turned off; and *ii)* if the natural light is at least lux_2 (gte_lux_2), then the lights must be turned off. The above properties are represented as follows:

Economy rules

$e_1 :$ $gte\_lux_1 \wedge (chosen\_lights \vee default\_lights) \rightarrow no\_lights'$
$e_2 :$ $gte\_lux_2 \rightarrow no\_lights'$

Now, consider the following scenario. On a bright Summer's day, John is working in his office when suddenly the fire alarm goes off. He leaves the office immediately. Once outside the building, he realises that he left his briefcase behind and decides to go back to fetch it. By the time he enters his office, the alarm has been going off for more than T_3 minutes. This situation can be formalised as follows:

i_1: John enters the office ($user\_in$)
i_2: The alarm is sounding ($alarm$)
i_3: T_3 minutes or more have elapsed since the alarm went off ($elapsed\_T_3$)
i_4: Day light provides luminosity enough to dispense with artificial lighting (gte_lux_2)

We get inconsistency in two different ways:

1. Because John walks in the office (i_1), the default light setting is chosen (r_7). By s_4, the lights must be on in this setting. This is a contradiction with safety rule s_2, which states that lights should be turned off T_3 minutes after the alarm goes off.
   $user\_in$ (i_1), $alarm$ (i_2), $elapsed\_T_3$ (i_3)
   $default\_lights' \rightarrow \neg no\_lights'$ (s_4)
   $user\_in \rightarrow default\_lights'$ (r_7)
   $alarm \wedge elapsed\_T_3 \rightarrow no\_lights'$ (s_2)
2. Similarly, when John walks in the office (i_1), the default light scene is set (r_7). This effectively

forces the lights to be turned on (s_4). However, by e_2, this is not necessary since the amount of luminosity coming from outside is higher the the maximum luminosity achievable by the office lights (gte_lux_2).

$user\_in$ (i_1), gte_lux_2 (i_4)
$default\_lights' \rightarrow \neg no\_lights'$ (s_4)
$user\_in \rightarrow default\_lights'$ (r_7)
$gte\_lux_2 \rightarrow no\_lights'$ (e_2)

We are, therefore, in a situation where inconsistency on the light scenes occur due to a safety property violation and due to an economy property violation. We need to reason about the courses of action to deal with this problem. Using clustered belief revision, we can arrange the several components of the specification in different priority settings, by grouping rules in clusters, e.g., safety cluster, economy cluster, etc. The organisation of the information in each cluster can be done independently but the overall prioritisation of the clusters at the highest level requires input from all stakeholders. Since the specification is being refined, the framework must cope with potential inconsistencies without trivialising the results. The formalism allows for arbitrary orderings *inside* the clusters as well, but this is not considered here for reasons of space and simplicity.

For example, in the scenario described previously, we might wish to prioritise safety rules over the other rules of the specification and yet not have enough information from stakeholders to decide on the relative strength of economy rules. In this case, we would ensure that the specification satisfies the safety rules but not necessarily the economy or ones.

Let us assume that sensor and factual information is correct and therefore not subject to revision. We combine this information in a cluster called "update" and give it highest priority. In addition, we assume that safety rules must have priority over economy rules. At this point, no information on the relative priority of behaviour rules is available. With this in mind, it is possible to arrange the clusters with the update, safety, behaviour and economy rules as depicted in Figure 2.[3] Prioritisations L1, L2 and L3 represent all possible linear arrangements of these clusters with the assumptions mentioned above, whereas prioritisations P1 and P2 represent the corresponding partial ones. As we mentioned, each of the components economy, behaviour, safety and update could be associated with its own partial priority order as well, allowing for the expression of more complex relationships between individual properties.

The overall result of the clustered revision will be consistent as long as the cluster with the highest priority (factual and sensor information) is not itself inconsistent. When the union of the sentences in the clusters is indeed inconsistent, in order to restore consistency, some rules may have to be withdrawn. The result will be such that rules will be kept as long as their inclusion does not cause inconsistency with other rules in a cluster with higher priority. Note that, to check whether the revised specification satisfies a rule, one needs to check for derivability of that rule from the final result.

For example, take prioritisation L1. The sentences in the safety cluster are consistent with those in the update cluster; together, they conflict with behaviour rule r_7 (see Figure 3).

Since r_7 is given lower priority in L1, it cannot be consistently kept and it is withdrawn from the intermediate result. The final step is to incorporate what can be consistently accepted from the economy cluster, for example e_2.[4]

Notice however, that r_7 might be kept given a different arrangement of the priorities. The refinement process occurs by allowing one to reason about these different arrangements and the impact of rules in the current specification without trivialising the results. Eventually, one aims to reach a final specification that is consistent regardless of the priorities between the clusters, i.e. in the classical logic sense, although this is not essential in our framework.

Prioritisations L2 and P2 give the same results as L1, i.e. withdrawal of r_7 is recommended. On the other hand, in prioritisation L3, the sentence in the behaviour cluster is consistent with those in the update cluster; together, they conflict with safety rule s_4 (see Figure 4).

Since the safety cluster is given lower priority in L3, both sentences s_2 and s_4 cannot be consistently kept. One has to give up either s_2 or s_4. However, if s_4 were to be kept, then e_2 would also be required to be withdrawn. The only way to cause minimal change to the specification is therefore to keep s_2 instead, since it allows the inclusion of e_2.

Finally, prioritisation P1 offers a choice between the sets of clusters {update, safety, economy} and {update, behaviour, economy}. The former corresponds to withdrawing r_7 reasoning in the same way as for L1, L2 and P2, while the latter corresponds to withdrawing s_4 as in the case of L3. It is not possible to make a choice based on the available priority information and hence the disjunction of results 1 and 2 above is taken.

In summary, from the five different cluster prioritisations analysed, a recommendation was made to withdraw a behaviour rule in three of them, to withdraw a safety rule in one of them, and to withdraw either a behaviour or a safety rule in one of them. From these results and the LCS context, the withdrawal of behaviour rule r_7 seems more plausible. In more complicated cases, a decision support system could be used to help the choice of recommendations made by the clustered revision framework.

4. RELATED WORK

A number of logic-based approaches for handling inconsistency and evolving requirements specifications have been proposed in the literature. Zowghi and Offen [20] proposed belief revision for default theories as a formal approach for resolving inconsistencies. Specifications are formalised as default theories where each requirement may be defeasible or non-defeasible. Each type is assumed to be consistent. Inconsistencies introduced by an evolutionary change are resolved by performing a revision operation over the entire specification. Change actions for handling inconsistency are implicitly given by the definition of such a belief revision operator, which changes the status of information from defeasible to non-defeasible and vice-versa to remove the inconsistent. Non-defeasible information that is inconsistent with defeasible information is

[3]Recall that a connecting arrow between clusters indicates priority of the source cluster over the target one.

[4]e_1 is also implicitly incorporated since we can neither prove the antecendent nor the negation of the consequent.

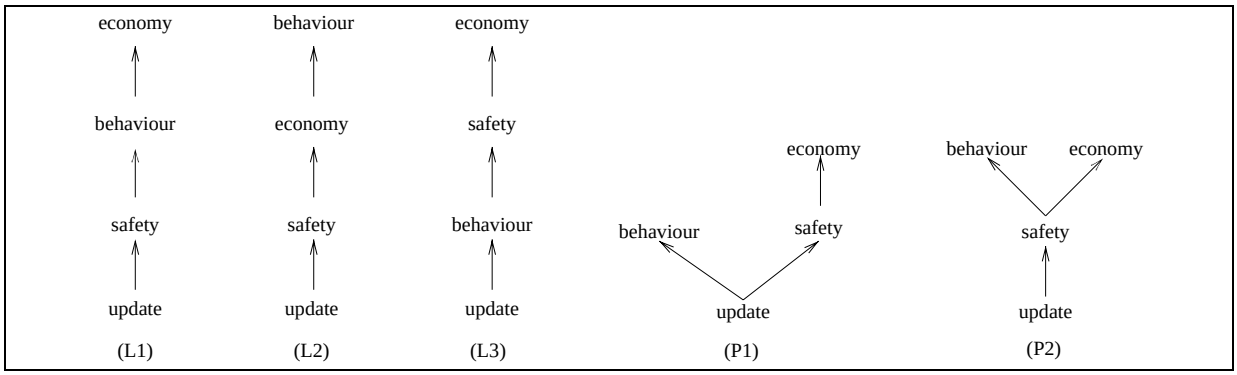

Figure 2: Linearly (L1, L2 and L3) and partially (P1 and P2) ordered clusters.

update + safety include (in DNF): $user\_in \wedge alarm \wedge elapsed\_T_3 \wedge gte\_lux_2 \wedge no\_lights' \wedge \neg default\_lights'$
behaviour includes (in DNF): $\neg user\_in \vee default\_lights'$
result 1: $user\_in \wedge alarm \wedge elapsed\_T_3 \wedge gte\_lux_2 \wedge no\_lights' \wedge \neg default\_lights'$

Figure 3: Conflict with behaviour rule r_7.

not taken into consideration during the reasoning process (thus avoiding trivialisation). Similarly, in our approach, requirements with lower priority that are inconsistent with requirements with higher priority are not considered in the computation of the revised specification. However, in our approach, the use of different levels of priority enables the engineer to fine-tune the specification and reason with different levels of defeasibility.

In [18], requirements are assumed to be defeasible, having an associated *preference ordering relation*. Conflicting defaults are resolved not by changing the specification but by considering only scenarios or models of the inconsistent specification that satisfy as much of the preferable information as possible. Whereas Ryan's preference relation is similar to our priority relation, the use of clusters in our approach provides the formalisation of the requirements with addtional dinemsions, which enables a more refined reasoning process about the inconsistencies.

In [4], a logic-based approach for reasoning about requirements specifications based on the construction of goal tree structures is proposed. Analyses of the consequences of alternative changes are carried out by investigating which goals would be satisfied and which would not, after adding or removing facts from a specification. In a similar fashion, our approach supports the evaluation of consequences of evolutionary changes by checking which requirements are lost and which are not after adding or deleting a requirement. Priority plays an important role in this process as the analysis could be focused on those requirements that have the highest priority only.

Finally, many other techniques have been proposed in the literature on managing inconsistency, but much of this work has focused on consistency checking, analysis and action based on pre- defined inconsistency handling rules. For example, in [5], consistency checking rules are combined with pre-defined lists of possible actions, but with no policy or heuristics on how to choose among alternative actions. The entire approach relies on taking decisions based on an analysis of the history of the development process (e.g., past inconsistencies and past actions). Differently, our approach provides a formal support for analysing the impact of changes over the specification by allowing the engineer to perform *if* questions on possible changes and to check the effect that these changes would have in terms of requirements that are lost or preserved.

5. CONCLUSIONS & FUTURE WORK

In this paper, we have shown how clustered belief revision can be used to analyse the results of different specification prioritisations reasoning classically, and to evolve specifications that contain conflicting viewpoints in a principled way.

We developed a tool for clustered revision and used a simplified version of the light control case study to provide an early validation of the tool. We believe that this approach provides the engineer with more freedom to make appropriate choices on the evolution of the requirements, while at the same time offering rigourous means for evaluating the consequences that such choices have on the specification.

Our approach is not only a technique for revising requirements specifications using priorities, but also a methodology for handling evolving requirements. The emphasis of the work is on the use of priorities for reasoning about potentially inconsistent specifications. The same technique can be used to check the consequences of a given specification and to reason about "what if" questions that arise during evolutionary changes.

One of the main issues in any existing and new formal reasoning technique for requirements engineering is the scalability problem. We believe that there is no universal solution to this problem and that it should be looked into and tackled on a case by case basis. Requirements for a given software system might for instance be formalised as a ground first-order logic theory, in order to make the reasoning process decidable. In our approach, a number of heuristics about the behaviour of the ordering $\preceq$ have been investigated. The use of DNF greatly simplifies the reasoning, but the conversion to DNF sometimes generates complex formulae. One possibility currently under research is the

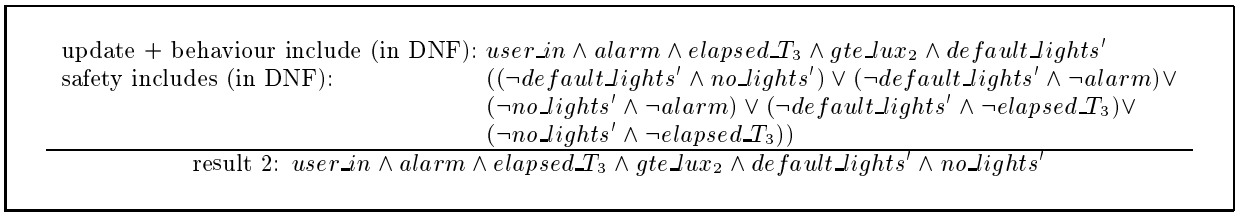

Figure 4: Conflict with safety rule s_4.

use of Karnaugh maps in order to find "minimal" DNF representations of the sentences.

The work described in this paper pressuposes the existence of a prioritisation theory for requirements specification. How to prioritise the requirements is a complex issue and it is outside of the scope of the research covered in this paper. We intend to look at these issues in the future.

Also, we intend to apply different Machine Learning [13] techniques to revise requirements specifications [7, 12]. Consider, for instance, the Light Control example of Section 3. Assume that a person in a particular office needs to have a light scene that violates the economy properties of the specification. This is a *scenario* which, in terms of Machine Learning, can be seen as an example to be learned. This example, when trained, may evolve the specification into a consistent new specification. In fact, Machine Learning techniques may add new concepts to the specification, according to the scenarios available for training. Differently from Belief Revision, though, Machine Learning techniques do not normally guarantee consistency of the new specification, nor that the principle of Minimal Change is satisfied. A comparative analysis of these two methods of theory revision in the context of requirements evolution would be highly desirable.

6. REFERENCES

[1] C. A. Alchourrón and D. Makinson. On the logic of theory change: Contraction functions and their associated revision functions. *Theoria*, 48:14–37, 1982.

[2] N. D. Belnap, A useful four-valued logic. Modern Uses of Multiple-Valued Logic, eds. G. Epstein and J. M. Dunn, Reidel Publishing Company, pp. 7-37, 1977.

[3] N. C. A. da Costa, On the theory of inconsistent formal systems. Notre Dame Journal of Formal Logic, 15(4):497–510, 1974.

[4] D. Duffy, C. MacNish, J. McDermid and P. Morris, A Framework for Requirements Analysis Using Automated Reasoning, Proceedings of CAiSE95, LNCS 932, Springer, 68-81, 1995.

[5] S. Easterbrook and B. Nuseibeh, Using ViewPoints for Inconsistency Management. In Software Engineering Journal, 11(1): 31-43, BCS/IEE Press, January 1996.

[6] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh, Inconsistency handling in multi-perspective specifications, IEEE Transactions on Software Engineering, 20(8), 569-578, 1994.

[7] A. S. d'Avila Garcez, A. Russo, B. Nuseibeh and J. Kramer. Combining Abductive Reasoning and Inductive Learning to Evolve Requirements Specifications. IEE Proceedings - Software 150(1):25-38, 2003.

[8] Peter Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. A Bradford Book - The MIT Press, Cambridge, Massachusetts - London, England, 1988.

[9] C. Heitmeyer and R. Bharadwaj, Applying the SCR Requirements Method to the Light Control Case Study, Journal of Universal Computer Science, Special Issue on Requirements Engineering: the Light Control Case Study, Vol.6(7), 2000.

[10] D. Gabbay and A. Hunter, Making inconsistency respectable 1: A logical framework for inconsistency in reasoning, Foundations of Artificial Intelligence Research, eds. Ph. Jorrand and J. Kelemen, LNCS 535, pp. 19-32, Springer, 1991.

[11] M. R. Huth and M. D. Ryan. Logic in Computer Science: Modelling and Reasoning about Systems. 387 pages. Cambridge University Press, 2000.

[12] A. van Lamsweerde and L. Willemet, Inferring Declarative Requirements Specifications from Operational Scenarios, IEEE Transactions on Software Engineering, Special Issue on Scenario Management, 1998.

[13] T. Mitchell, Machine Learning, McGraw Hill, 1997.

[14] B. Nebel. Syntax based approaches to belief revision. *Belief Revision*, pages 52–88, 1992.

[15] B. Nuseibeh, J. Kramer and A. Finkelstein, A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification, IEEE Transactions on Software Engineering, 20(10): 760-773, October 1994.

[16] S. Queins et al., The Light Control Case Study: Problem Description. Journal of Universal Computer Science, Special Issue on Requirements Engineering: the Light Control Case Study, Vol.6(7), 2000.

[17] O. Rodrigues, Structured Clusters: A Framework to Reason with Contradictory Interests, Journal of Logic and Computation, 13(1):69–97, 2003.

[18] M. D. Ryan. Default in Specification, IEEE Proceedings of International Symposium on Requirements Engineering (RE93), 266-272, San Diego, California, January 1993.

[19] G. Spanoudakis and A. Zisman. Inconsistency Management in Software Engineering: Survey and Open Research Issues, Handbook of Softawre Engineering and Knowledge Engineering, (ed.) S.K. Chang, pp. 329-380, 2001.

[20] D. Zowghi and R. Offen, A Logical Framework for Modeling and Reasoning about the Evolution of Requirements, Proc. 3rd IEEE International Symposium on Requirements Engineering RE'97, Annapolis, USA, January 1997.

The Impacts of Software Design on Development Effort – a Differential Evolution Approach

Päivi Ovaska
Teaching researcher
Lappeenranta University of Technology
Department of Information Technology
P.O. Box 20 53851 Lappeenranta
Finland
+358 40 5339231
paivi.ovaska@lut.fi

Alexandre Bern
Researcher
Lappeenranta University of Technology
Department of Information Technology
P.O. Box 20 53851 Lappeenranta
Finland
bern@lut.fi

ABSTRACT
In this study we analyzed the relationship between software design and development effort (development time) in a real life software project by using a novel approach called differential evolution. The two subsystems developed separately in two subprojects were analyzed to make comparisons between these two. It was found out that coupling between modules had the greatest influence on the software development effort in the situation, where human and organizational factors in software development were strongly present. In the other subsystem, where the whole development work was handled better, the size measure was the most important factor that affected the software development effort. These results are much line with the other research results, where software size as well as cohesion and coupling are reported the most important factors affecting to the cost of software development. The results of this study also suggest that differential evolution approach is suitable for analysing the relationships between software design and development effort.

Keywords
Software design, differential evolution, case study

1. INTRODUCTION
It is well known that software systems typically exceed their estimated development costs. There are many factors that can affect exceeding of development costs in software project. These factors include system structural factors as well as human and organizational factors [1], [2]. One of the most important factors reported in the research is the size of the system to be developed [3], [4], [22], [23]. Many measures of the software size ranging from the number of lines of the source code to functional size measures such as function points have been proposed [5]. Size can be measured in various ways at different phases of the software development ranging from the requirements analysis phase to the coding phase [6]. In addition to size, many other properties such as cohesion and coupling (telling the complexity of a system) have been mentioned as the cost factors [7], [8].

The work described in this paper had several different but interconnected objectives. First, we wanted to better understand the relationship between software design and the development effort (development time) from structural as well as human and organization point of views. We wanted to find out which design properties including size, coupling and cohesion have a clear relationship to the development effort. Second, we wanted to get experience on gathering needed data from the design phase described in design specification documents. We were interested in the limitations of industrial design specifications and information that could be extracted from these specifications. Third, we wanted to get experience of using differential evolution (DE) in analyzing the relationships between software design and development effort.

The next part of this paper (Section 2) describes the research settings. In section 5 is explained the research subject and methods. Section 4 explains the results of this study. In Section 5, we discuss the research results and topics for the further study.

2. RESEARCH SETTINGS
2.1 Design metrics
The main goal in developing the metrics was to create a set of metrics that would characterize our system best based on the experience from the project development. We also wanted these metrics to be gathered as much as possible from the design phase. This was our second goal. The third goal was to create metrics that would be as independent on each other as possible.

The metrics used in our study and related to them attributes are listed in Table 1. All these metrics are directed to single modules, not to entire subsystems. The subsystems are evaluated on the basis of the values of the attributes of the metrics.

Among all definitions around coupling and cohesion [9], [10], [2], we used the following basis to these metrics in our study:

Coupling. According to [11], pp. 375, coupling can be defined as follows: “Coupling is a measure of interconnection

among modules in a program structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface." During the analysis and design phase, we can measure intermodular coupling by the number of relationships between the subsystems [2], pp.110. According to this definition, coupling measures the amount of interconnections (references) between modules. Here, two different metrics for coupling are used. Attribute a_3 refers to the number of modules to which the module being studied refers, whereas attribute a_4 defines the number of modules referring to the module being studied.

Cohesion. According to [11], pp. 374, cohesion is "a measure of the relative functional strength of a module." Within the limits of this project, cohesion is defined as a number of aggregations, compositions and relations in the class diagram of a module. Stronger cohesion should be achieved in order to implement an internally strong module.

Table 1. Attributes (design properties)

Attribute	*Metric name*	*Description*
a_1	KLOC	Number of **K**ilo **L**ines **O**f **C**ode in a module
a_2	NOC	**N**umber **O**f **C**lasses in a module
a_3	Coupling 1	Number of modules referring to this module
a_4	Coupling 2	Number of modules this module refers to
a_5	Cohesion	Number of aggregations, compositions and relations among classes of a module
a_6	NOUC	**N**umber **O**f **U**se **C**ases of a module
a_7	NOS	**N**umber **O**f **S**ubmodules forming a module
a_8	NOD	**N**umber **O**f **D**atabases connected to a module

2.2 The studied system

The system studied was implemented in a Finnish telecommunications company and consists of two subsystems, a CORBA–based (Common Object Request Broker Architecture), highly distributed server (let it be subsystem B) and a centralized client (let us call it subsystem A).

The user requests a service through the user interface that can be either mobile or World Wide Web (WWW). The service fetches the requested information, processes it and returns the reply to the user.

The platform was mainly designed for the needs of the international market and has to support interfaces for different kinds of external systems, e.g. SMS (Short Message) centers, WAP (Wireless Application Protocol) gateways, other platforms, billing systems etc. The needs of the international market posed additional demands on the user interfaces; for example, they should be localizable to any language and should show the results in a country-specific manner. This also posed challenges on subsystem B: it has to be possible to dissipate the information all over the world, different kinds of information protocols must be supported and so on.

Subsystem A is responsible for the user interfaces, authentication and authorization as well as for the interfaces to external systems. It converts the end-user's request to a standard request for subsystem B, replies to the standard reply from subsystem B and sends the reply back to the end-user. Subsystem A does not know the location of the information maintained by subsystem B or how that information is retrieved from the information resources all over the world. This requirement was not well implemented; subsystem A was dependent on the types of information resources residing in subsystem B.

Subsystem B is responsible for dynamically resolving the information resources to be used by examining the request and routing it to the right information resource. To do that, subsystem B uses CORBA (Common Object Request Broker) Trading Service. The main requirement for subsystem B is high configurability: new information resources and services should be added to it by simply configuring it and adding the new modules to the system. Subsystem B is geographically distributed. The distribution is implemented using CORBA technology.

2.3 Data collection

We collected two separate subsets of data: one for subsystem A and the other one for subsystem B. These data were later used to compare the subsystems. Based on the specification documents, we succeeded to define the number of sub-modules and databases as well as the coupling and cohesion for some of the modules of both subsystems. The rest of the information was re-engineered from the implementation codes.

Some of the specification documents were not up-to-date, which made it necessary to study the implementation codes more carefully. The UML diagrams turned out to be unreliable for some modules; they were re-engineered using the Together 5.5 development tool for application modeling and round-trip engineering for Java and C++ [12].

The numbers of lines of codes were obtained by using an application for counting the lines of code downloaded from the Web [13]. When counting the numbers of lines, comments were ignored.

The extracted values of the attributes are shown in Table 2 for subsystem B and Table 3 for subsystem A, respectively. The

values that define the development times were taken from the project management software (Niku Workbench).

In the Table 2 and Table 3, we have corrected development time of some modules according to knowledge about the heterogeneous professional competence of some developers and the assumption that it has a strong effect to development effort [14]. In subsystem B (Table 3), only the development time for one module is corrected (by dividing the time by 0.76), whereas for subsystem A (Table 4), the development times are corrected for all six modules. The corrected coefficients were based on the COMOMO II PCAP Cost Driver factor (PCAP, Programmer Capability) in [14], pp.48.

Table 2: Values of the attributes of subsystem B.

Attribute	A_1	A_2	A_3	A_4	A_5	A_6
a_1	1	7	3	4	1	1
a_2	9	53	43	47	23	10
a_3	4	2	1	4	3	1
a_4	4	2	4	1	0	3
a_5	5	65	30	21	9	10
a_6	10	7	12	3	13	7
a_7	1	2	1	1	1	2
a_8	0	4	1	1	0	0
Uncorrected development time (h)	540.5	634.5	889.5	712	417	579
Correction coefficient	1.0	0.76	1.0	1.0	1.0	1.0
Corrected development time (h)	540.5	835	889.5	712	417	579

Table 3: The values of the attributes of subsystem A.

Attribute	B_1	B_2	B_3	B_4	B_5	B_6
a_1	1	6	1	2	10	3
a_2	20	13	3	8	118	14
a_3	2	0	1	1	5	5
a_4	2	5	2	3	2	1
a_5	6	9	0	0	10	9
a_6	19	6	8	7	17	3
a_7	1	3	1	1	4	1
a_8	0	0	0	0	1	1
Uncorrected development time (h)	1220	1488	934	950	966	1141
Correction coefficient	1.15	1.15	1.15	1.15	0.76	1.15
Corrected development time (h)	1061	1294	812	826	1271	993

3. RESEARCH SUBJECT AND RESEARCH METHOD

The aim of this study was to analyze the impact of software design on the development effort in an industrial project. We created the design metrics that characterized our system best using the practical experiences from project. This experience showed that there were problems with module integration especially in subsystem A. This suggested to us that there were problems with interfaces between modules. Based on this knowledge, we created the hypothesis for our study: coupling impacts mostly on the development effort. This hypothesis is to be proved in this study.

The development effort in this study refers to the effort needed for the design, implementation and module testing of a module.

To study the impact of design on development effort we considered two novel methods: a non-linear global optimization method called Differential Evolution Algorithm (DEA) and a modeling method based on an Artificial Neural Network (ANN) [16], [17], [18], [19]. The insufficient amount of data (only six modules per subsystem) made it impossible to use the latter approach and gave us an opportunity to try a novel but already popular and widely used approach to global optimization, Differential Evolution (DE).

Traditional optimization methods, such as exhaustive search, analytical optimization and the Simplex method [20] were not considered because of the trickiness of the objective function: trickiness is based on a difficult structure obtained by combining several equations (each module has its own equation that depicts the model as shown in the equation) into one objective function. The other reason is that there are restrictions (in intervals) involved for optimization.

Based on the metrics suite, the development effort of the subsystems' modules was estimated by using a linear model containing the attributes of the metrics suite as the variables. The linear model was selected for the reason that it is given by a simple D-dimensional function formed by a sum of variables and related to them linear coefficients, but it still has the capability of estimating the development effort with sufficient precision. The model is given by equation (1).

$$H(x_1, x_2, ..., x_8) = b_1 x_1 + b_2 x_2 + ... + b_8 x_8 \quad (1)$$

The variables of the above equation refer to the attribute of the metrics suite in such a way that variable x_1 corresponds to attribute a_1 and variable x_8 corresponds to attribute a_8, respectively. The coefficients of the model labelled by b_k (k = 1…8) define the significance rate of the corresponding attributes. The attributes are thought to be significant if the values of their coefficients are positive, which cause them to influence on the value of the function. When the values of the coefficients are known, assigning the corresponding values to the variables of the model gives the development effort of the corresponding module.

The values of the coefficients of the model were defined by minimizing the corresponding objective function by the DEA. The objective function is given by equation (2).

$$W(b_1, b_2, ..., b_n) = \frac{1}{m}\sum_{m=1}^{6}\left(100\frac{|H_m - h_m|}{h_m}\right) = \frac{10^2}{m}\sum_{m=1}^{6}\frac{|H_m - h_m|}{h_m} \quad (2)$$

In the above equation,

b_n	The coefficient of the n^{th} attribute
m	The number of the modules of a subsystem
H_m	The value of the cost estimation function (equation (1)) for the module referred by m.
h_m	The value of the measured development effort of the module referred by m.

As a result, W returns the mean error between the measured and estimated development effort of all six modules. For some modules, we adjusted coefficients b_k (k = 1…8), which are defined for the cost estimation function of the subsystem's modules in order to minimize the value of W. In the ideal case, the value of the objective function is zero, which means that the cost estimation function returns the same value as the value of the measured development effort for all six modules of a subsystem.

To minimize the objective function, the two following DE schemes were used: DE/best/1/bin [15] and DE/rand/1/bin [21]. Since the schemes produced the same results, they are given only once.

4. RESEARCH RESULTS

In this section we describe the results of the DE analysis. The analysis is divided into two parts: identification of significant attributes (the attributes greater than zero) and measuring the importance of those attributes. Once the significant attributes are identified, they are measured for their importance. A significant attribute is thought to be important if, when excluded from the model, it causes an increase to a certain degree in the value of the objective function when optimized again. A high degree of increase means that the model is not capable of fitting the data well without the excluded attribute thus making that attribute very important. The most affective attributes are defined through the combined use of the significance and importance measures.

4.1 Identification of significant attributes

Table 4 shows the values of the coefficients of the model, which were obtained by minimizing the objective function when taking all six design properties (i.e. attributes). The table illustrates that for subsystem B the significant attributes are a_2 (NOC), a_3 (coupling 1), a_4 (coupling 2), a_6 (NOUC), and a_7 (NOS), whereas for subsystem A, these attributes are a_3 (coupling 1), a_4 (coupling 2), a_5 (cohesion), and a_6 (NOUC), respectively, since the values of the corresponding coefficients are greater than zero. The last row of the table contains the values of the objective function, which is the mean percentile error between the measured and estimated development effort of all six modules.

As the Table 4 shows, for both subsystems, the coefficient values differ significantly from each other. Some design properties that influence the development effort of subsystem A have no influence on the development effort of subsystem B and vice versa.

Table 4: The values of the coefficients of the model

Coeff.	*Values of subsystem A*	*Values of subsystem B*
b_1	0.0	0.0
b_2	0.0	10.5
b_3	110.0	4.8
b_4	200.5	68.8
b_5	21.5	0.0
b_6	16.4	3.1
b_7	0.0	120.6
b_8	0.0	0.0
W	5.8	2.6

4.2 Measuring the importance of the attributes

We measured the importance of each attributes by excluding each attribute in turn from the model. Now, the rest of the attributes got different values producing different object function results as presented in Table 5.

Table 5: A summary of the mean errors between the measured and estimated development effort

Excluded attribute	*W (subsystem A)*	*W (subsystem B)*
none	5.8	2.6
b_1	N/A	N/A
b_2	N/A	10.5
b_3	8.1	3.0
b_4	**28.1**	**12.3**
b_5	6.7	N/A
b_6	8.5	3.8
b_7	N/A	5.3
b_8	N/A	N/A

N/A in the table above means that the corresponding attribute is not significant (please refer to Table 4). The increase in the error is illustrated in Table 6.

Table 6: The value of the increase in the error (%) while excluding significant attributes one by one from the model

Excluded attribute	*The increase in the error in subsystem A*	*The increase in the error in subsystem B*
a_1	-	-
a_2	-	299%
a_3	41%	44%
a_4	297%	367%
a_5	15%	-
a_6	-	-
a_7	-	101%
a_8	-	-

4.3 Analysis of the results

The results suggest that the most important attribute is a_4 (coupling 2). When excluding this attribute from the model, the error increases up to 297% for subsystem A and 367% for subsystem B (Table 6). This means that attribute a_4 has a very strong correlation with the development effort. The value of a_4 was also very high for both subsystems. Coupling 1 (attribute a_3) also clearly correlates with the development error for both subsystems and gets high values especially in the case of subsystem A.

In subsystem B, attributes a_2 (NOC, number of classes) and a_7 (number of submodules) had a strong correlation with development effort as well. And again, in the analysis of subsystem A, cohesion (attribute a_4) showed some correlation with the development effort.

Human and organizational aspects of the software development in these projects can explain these different results of analysis of the subsystems. Subsystem A was implemented within the same site by experienced developers who used prototyping to design the interfaces between the modules. Subsystem B was developed in different sites by less experienced software engineers. They did not use any prototyping to help interface design in this subsystem, and the developers confronted sizeable problems when integrating the modules, because the interfaces between them had not been properly designed.

5. CONCLUSIONS

This study focused on analyzing the relationships between software design and development effort in a single industrial software project. We created a design metrics suite that characterized our practical system best and gave us possibility to collect data mostly from design phase. We wanted to understand system structural properties as well as human, organizational and process factors. To create the metrics suite, we used our experiences obtained from the system development. Based on this metric suite, we defined the coefficients of the model estimating the development effort of the system under the study using a global non-linear optimization method, a differential evolution algorithm.

We found out in our study that coupling between the modules was the most important design property that affected the software development effort in the situation where human and organizational factors in software development were strongly present. In our study, these factors were lack of competence of software developers, poor coordination of the development work and poor designing of interfaces between the modules. In the other subsystem, where development was better managed, the size measure (number of classes in the module) affected mostly the development effort. These results are much line with research results, where software size [3], [4], [22], [23] and complexity metrics (coupling and cohesion) [7], [8] have reported as the most important factors affecting on the software costs. We also noticed that software design specifications were not up-to-date in our project and we had to using re-engineering in order to get some metrics information from the source code.

The results of this study also suggest that the differential evolution approach is suitable for analyzing software development effort and encourage us to analyze other industrial projects too. In the scope of this study, it seems that the attributes of the design metrics suite used are not completely independent. Future research could focus on studying the interdependences of the design attributes. The results show that excluding specific attribute from the model had an influence on the values of the coefficients of the other attributes as well as of the result. This phenomenon remains to be interpreted. Increasing the value of a specific significant attribute by, for instance, one percent and studying how this change affects the other significant attributes can perform local sensitivity analysis of the attributes.

6. REFERENCES

[1] B. Bruegge & A. H. Dutoit, Object-Oriented Software Engineering: Conquering Complex and Changing Systems. Prentice Hall, 2000, ISBN: 0-13-489725.

[2] B. Henderson-Sellers, Object-Oriented Metrics: Measures of complexity, Prentice Hall, New Jersey, 1996. pp 39-40

[3] L.Briand, J.Daly, V.Porterm, J. Wüst,"Exploring the relationshiops between design measure and software quality in object-oriented systems", Journal of Systems and Software 51, p. 245-273, 2000.

[4] Lionel C. Briand, Jürgen Wüst, "The Impact of Design Properties on Development Cost in Object-Oriented Systems",Software Metrics Symposium, 2001, METRICS 2001, 7th International Proceedings on Software Metrics, London, UK. 4-6 April 2001, Pages: 260 – 271, ISBN: 0-7695-1043-4.

[5] H.D Rompach, "Design Measurement: Some Lessons Learned" , IEEE Software, pp.17-25, March 1999

[6] N.Fenton, S.L. Pfleeger, Software Metrics. A Rigorous and Practiclal Approach. International Thomson Computer Press, London 1997.

[7] L. Briand, J. Daly, J. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems, IEEE Transactions on Software Engineering 25 (1), 91-22, 1999

[8] L. Briand, J. Daly, J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems, Empirical Software Engineering Journal, 3(1), 1998

[9] D. P. Darcy, C.F. Kemerer, "Software Complexity: Toward a Unified Theory of Coupling and Cohesion, IDSc Workshop, Management Information Research Center, Spring 2002, http://misrc.umn.edu/workshop/spring2002/Darcy_020802.pdf

[10] Franck Xia, "Module Coupling: A Design Metric", Proceedings of the 1996 Asia-Pacific Conference on Software Engineering. Seoul, South Korea, 4-7 Dec. 1996, Pages: 44 – 54, ISBN: 0-8186-7638-8.

[11] Roger S. Pressman, Software Engineering, A Practitioner's Approach, 4th edition, The McGraw-Hill Companies, Inc.. ISBN: 0077094115, p. 129.

[12] The official Web-site of the TogetherSoftTM Corporation: TogetherSoft, a Development Tool for an Application Modeling and Round Trip Engineering for Java and C++. Web-document, URL: http://www.togethersoft.com/. [Referred 22.09.2002].

[13] Lines of Code Counter, Dr. John Dalbey's Web-page on Web-server of Dep. of Comp. Sc., California Polytechnic State University,Web-document, URL: http://www.csc.calpoly.edu/~jdalbey/SWE/PSP/LOChelp.html; [Referred 22.09.2002].

[14] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer and Bert Steece, Software Cost Estimation with COCOMO II, Prentice-Hall, Inc., 2000, ISBN: 0-13-02-6692-2.

[15] Rainer Storn, "On the Usage of Differential Evolution for Function Optimization" (Editors: Smith, M.H., Lee, M.A., Keller, J., Yen), 1996 Biennial Conference of the North American. Fuzzy Information Processing Society, 1996. NAFIPS. Berkeley, CA, USA. 19-22 June 1996. Pages: 519 – 523. ISBN: 0-7803-3225-3.

[16] Ali Idri, Taghi M. Khoshgoftaar, Alain Abran, "Can neural Networks be easily Interpreted in Software Cost Estimation?", Proceedings of the 2002 IEEE International Conference on Fuzzy Systems,FUZZ-IEEE'02, Honolulu, HI, USA, 12-17 May 2002, Volume: 2, Pages: 1162 – 1167, ISBN: 0-7803-7280-8.

[17] Gavin R. Finnie and Gerhard E. Wittig, "AI Tools for Software Development Effort Estimation" (Editors: Purvis, M. Bond Univ., Gold Coast, Qld., Australia). Proceedings of the 1996 International Conference on Software Engineering: Education and Practice, Dunedin, New Zealand. 24-27 Jan. 1996, Pages: 346 – 353, ISBN: 0-8186-7379-6.

[18] A.R. Venkatachalam, "Software Cost Estimation Using Artificial Neural Networks", Proceedings of the 1993 International Joint Conference on Neural Networks, IJCNN '93-Nagoya, October 25-29, 1993, Volume 1, Pages: 987 – 990, ISBN: 0-7803-1421-2.

[19] W. Pedrycz, J.F. Peters, S. Ramanna, " A Fuzzy Set Approach to Cost Estimation of Software Project" (Editor: Meng, M.), IEEE Canadian Conference on Electrical and Computer Engineering, 1999, Edmonton, Alta, Canada, 9-12 May 1999, Volume: 2, Pages: 1068 – 1073, ISBN: 0-7803-5579-2.

[20] Juha Haataja, Optimointitehtävien ratkaiseminen. CSC – Tieteellinen laskenta Oy, 2. painos, 1995, in Finnish.

[21] Jouni Lampinen,"Multi-Constrained Nonlinear Optimization by the Differential Evolution Algorithm", A Document Proposing an Extension for the Differential Evolution Algorithm (DEA) for Handling Non-linear Constrain Functions, Web-document, URL: http://www.it.lut.fi/opetus/01-02/010778000/DECONSTR.PDF. [Referred 29.09.2002].

[22] Walston, C.E, P.C. Felix (1977), "A Method of Programming Measurement and Estimation", IBM Systems Journal, 55-73

[23] Halstead, Maurice H (1977), Elements of Software Science, Operating, and Programming

An Explanation Reasoning Procedure Applicable to Loop Transformation in Compiler

Mariko Sasakura
Graduate School of Natural Science and Technology, Okayama University
Tsushima-Naka 3-1-1
Okayama, 700-8530, Japan
sasakura@momo.it.okayama-u.ac.jp

Susumu Yamasaki
Graduate School of Natural Science and Technology, Okayama University
Tsushima-Naka 3-1-1
Okayama, 700-8530, Japan
yamasaki@momo.it.okayama-u.ac.jp

ABSTRACT

In this paper, we discuss a case that we apply the abductive procedure to some loop transformation in a parallelizing compiler. For the loop transformation, the compiler should investigate data dependences in loops. However, it is sometimes uncertain about whether there are data dependences or not. To cope with this uncertainty problem, we make use of the abductive procedure to present programmers the analyses of data dependences by the compiler and to assist some compiler construction. The procedure generates rules and declarations dynamically according to the output from the compiler, and infers the rules. The combination of parallelizing compilers and the abductive procedure may improve the intelligence of parallelizing compilers.

1. INTRODUCTION

In this paper, we discuss a case that we apply the abductive procedure to some loop transformation in a parallelizing compiler.

A parallelizing compiler transforms a sequential program to a parallel program. Parallelization of a sequential program is done by transforming sequential loops to parallel loops. There are many loop transformation methods that have been proposed [3]. A parallelizing compiler detects loops that can be executed in parallel and applies proper transformation methods to the loops.

We can detect whether a loop can be parallelized or not by checking data dependences of the loop : if there is no data dependence, the loop can be parallelized. However, the detection is not so easy, because the complete analysis of data dependences takes too much time for the compiler. Thus, a compiler may infer that some loops cannot be parallelized if it cannot confirm that there is no data dependence. This rule can be briefly represented as an extended logic program like the following (A more precise description is given in Section 3.):

```
      do 10 i = 1, 100
      A(i+1) = B(i)
      C(i) = A(i)
10    continue
```

Figure 1: A loop which has a data dependence

$$parallelize \leftarrow \neg dependence$$
$$\neg parallelize \leftarrow dependence$$
$$\neg parallelize \leftarrow\sim \neg dependence$$

where *parallelize* means the loop can be executed in parallel and *dependence* means there are data dependences in the loop. $\neg$ denotes explicit negation and $\sim$ denotes negation as failure. We will give brief explanation about them in Section 2.

A parallelizing compiler transforms sequential loops to parallel loops, because most of execution time of a program is spent for loops. The process of parallelization consists of two steps: data dependence analysis and loop transformation. The loops in which the compiler knows no data dependence, are transformed in parallel loops.

If there are more than one access to a variable or an element of an array in a loop, and the execution result is changed when the order of the accesses is changed, we call there is data dependence in the loop. For example, the loop in Fig. 1 has data dependence on the array A.

The existence of data dependence results in the existence of the integer number solution for equations which are constructed by subscripts of arrays in a loop. However, in practice, it takes too much time to solve the equations precisely. Therefore, compilers must check data dependences by simple test such as the GCD test [4] or the omega test [21]. These tests check evidences of no data dependence in short time. However, these tests cannot check all cases of no data dependence. In other words, there are cases that there is no data dependence but the tests cannot find them.

Even though a complier infers that a loop cannot be parallelized, in some case, there may be no data dependence in the loop. If we give programmers the explanation why the

compiler infers that the loop cannot be parallelized, they sometimes give more information such that "I know that there is no data dependence in the loop" and let the compiler know that the loop can be parallelized. Thus, we use the explanation reasoning procedure which is based on the abductive procedure with a parallelizing compiler. The procedure will give programmers the reason why the compiler infers the loop cannot be parallelized.

The outline of the explanation reasoning procedure for parallelizing compilers is as follows:

1. The compiler analyzes data dependences of loops.
2. If there are loops that cannot be parallelized, the explanation reasoning procedure generates an extended logic program from the analysis of the compiler and shows which part of data dependences are unknown.
3. Programmers give the data dependence information, if possible.
4. The compiler parallelizes the loops using the information.

Most dominant abductive procedures are shown and/or follow the interpretations as in [8, 11, 12, 13, 16, 33], which are based on negation as failure rule (as in [6, 17]), apart from the two-valued stable model in [9] and from the well-founded model in [20]. The procedure may be available even in distributed environments, as in [31]. The points of the present procedure are as follows.

- It is based on the abductive procedure for an extended logic program, while the extended logic program is expressive owing to two kinds of negation.
- It dynamically generates an extended logic program from the analysis of the compiler.
- Soundness and completeness of the procedure are related to model theory such that it can be implemented exactly by what is expressed and what the program means.

2. AN EXPLANATION REASONING PROCEDURE

Extending the procedure [7] applicable to general logic programs, we have a contradiction-free proof procedure, which is applicable to extended logic programs with capability of dynamically eliminating contradictory derivations. The contradiction-free proof procedure consists of two derivations, succeeding and failing derivations. It is sound with respect to semantics of the original program, if it is consistent. When the original program is inconsistent, its soundness is supported by semantics of the transformed program with exceptions, where the original program is transformed to the program with exceptions by introducing Kowalski-Sadri exceptions (as in [15]). This soundness differs from the soundnesses of the procedures in [1, 18, 19]. The semantics for the program with exceptions can be defined by means of the fixpoint of $\Gamma_s \bullet \Gamma_s$ in terms of the operator of [1], however, the relation between the expected procedure and the fixpoint (not always the least) is made clear in [32]. The semantics is based on model theory, while there is some rule priority techniques in [2].

An extended logic program (ELP, for short) is a set of clauses of the form:

$$L \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n \ (0 \leq m \leq n),$$

where L and L_i are literals (that is, atoms or their explicit negations), and "$\sim$" stands for the negation as failure. L of the clause is said to be its head, and $L_1, \ldots, L_m$, $\sim L_{m+1}, \ldots, \sim L_n$ its body. L and L_i are literals, where a literal is an atom A or its explicit negation $\neg A$ ([1, 9]). For an atom A, $\neg L$ means $\neg A$ if $L = A$, and $\neg L$ means A if $L = \neg A$. The pair of A and $\neg A$ is said to be complementary.

A goal is an expression of the form:

$$\leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n \ (0 \leq m \leq n),$$

where L_i are literals. The goal is denoted by $\Box$ if it contains no literal. The instance of a goal is an expression obtained from the goal by substituting some terms for variables.

A clause, a goal or a term is said to be a ground clause, a ground goal or a ground term, respectively, if it contains no variables.

In the ground version where variables are not contained in any clause and goal, the essential rules of the procedure are settled so that the following requirements are satisfied.

(1) (Contradiction elimination) In case that $\leftarrow l$ succeeds, l is in memory in a succeeding derivation so that l prevents $\leftarrow \neg l$ from succeeding.

(2) (Coherence principle) In case that $\leftarrow l$ succeeds, $\leftarrow \neg l$ fails.

(3) (Negation as failure) In case that $\leftarrow l$ fails with l (or $\sim l$) in memory, $\leftarrow \sim l$ succeeds. In case that $\leftarrow l$ succeeds, $\leftarrow \sim l$ fails.

To construct the procedure involving three cases as above shown, we prepare for

(a) the set of literals to be concerned with the succeeding derivation in the case of (1) (the suclit set, for short), to prevent resolutions from some succeeding derivations, and

(b) the set of ground literals to be concerned with the case of (3) for negation as failure (the naf set, for short).

In the procedure for an ELP Q, $M^{\sim}$ (in a goal) stands for a literal, or negation as failure for a literal. The explanation reasoing procedure consists of two derivations which are mutually recursive as follows. The soundness is to be guaranteed by some model defined by alternating fixpoint method (like [25, 26]) applied to extended logic programs with Kowalski-Sadri exceptions (as in [15]), while the program with exceptions is obtained by a little static transformation, and the proof procedure is implemented dynamically for the escape from contradictory derivations. The

formal theory is in [22, 32].

1. A succeeding derivation (suc):
A succeeding derivation from a goal G of length $h (h \geq 0)$ is a sequence

$$(G_0, \Sigma_0, \Delta_0), \ldots, (G_h, \Sigma_h, \Delta_h),$$

where $\Sigma_0, \ldots, \Sigma_h$ are suclit sets, $\Delta_0, \ldots, \Delta_h$ are naf sets and the sequence is organized by the following rules. The above derivation is denoted by

$$(G_0, \Sigma_0, \Delta_0) \Rightarrow_{suc} (G_h, \Sigma_h, \Delta_h).$$

When $G_h = \Box$, the derivation halts and we say that G_0 succeeds.

(Rules) Let $G_k = \leftarrow M_1^\sim, \ldots, M_n^\sim$, where $M_i^\sim$ is selected by the rule R. $(G_{k+1}, \Sigma_{k+1}, \Delta_{k+1})$ is obtained from $(G_k, \Sigma_k, \Delta_k)$ by:

(suc1) In case that there is $L \leftarrow N_1^\sim, \ldots, N_m^\sim \in P$ such that $M_i^\sim = L$, and there is no L' in Σ_k such that $L = \neg L'$,

$$G_{k+1} = \leftarrow M_1^\sim, \ldots, M_{i-1}^\sim, N_1^\sim, \ldots, N_m^\sim, M_{i+1}^\sim, \ldots, M_n^\sim,$$
$$\Sigma_{k+1} = \{M \mid M \in \Sigma_k\} \cup \{L\}.$$

(suc2) In case that $M_i^\sim = \sim L$ and $L \in \Delta_k$,

$$G_{k+1} = \leftarrow M_1^\sim, \ldots, M_{i-1}^\sim, M_{i+1}^\sim, \ldots, M_n^\sim,$$
$$\Sigma_{k+1} = \Sigma_k, \Delta_{k+1} = \Delta_k.$$

(suc3) In case that $M_i^\sim = \sim L$, $L \notin \Delta_k$, and there is a failing derivation $(\{\leftarrow L\}, \Sigma_k, \Delta_k \cup \{L\}) \Rightarrow_{ff} (\emptyset, \Sigma', \Delta')$,

$$G_{k+1} = \leftarrow M_1^\sim, \ldots, M_{i-1}^\sim, M_{i+1}^\sim, \ldots M_n^\sim,$$
$$\Sigma_{k+1} = \Sigma', \Delta_{k+1} = \Delta'.$$

2. A (finitely) failing derivation (ff):
For a set H of goals, a (finitely) failing derivation of length h $(h \geq 0)$ is a sequence

$$(H_0, \Sigma_0, \Delta_0), \ldots, (H_h, \Sigma_h, \Delta_h),$$

where $H_0 = H$, $\Box \notin H_k$ for $1 \leq k \leq h$, $\Sigma_0, \ldots, \Sigma_h$ are suclit sets, $\Delta_0, \ldots, \Delta_h$ are naf sets and the sequence is organized by the following rules. The above derivation is denoted by

$$(H_0, \Sigma_0, \Delta_0) \Rightarrow_{ff} (H_h, \Sigma_h, \Delta_h).$$

When $H_h = \emptyset$, the derivation halts and we say that each goal in H_0 fails.

(Rules) Assume that $H_k = H'_k \cup \{\leftarrow M_1^\sim, \ldots, M_n^\sim\}$, where $M_i^\sim$ is selected by the rule R in $\leftarrow M_1^\sim, \ldots, M_n^\sim$.

(ff1) In case that $M_i^\sim = L$:

(ff1-1) If L is a ground literal and there is a succeeding derivation

$$(\leftarrow \neg L, \Sigma_k, \Delta_k) \Rightarrow_{suc} (\Box, \Sigma', \Delta'),$$

then

$$H_{k+1} = H'_k, \Sigma_{k+1} = \Sigma', \Delta_{k+1} = \Delta'.$$

(ff1-2) If $\forall L' \in \Sigma_k : [L \neq \neg L']$, then

$$H_{k+1} = H'_k \cup \{G_1, \ldots, G_m\},$$
$$\Sigma_{k+1} = \Sigma_k, \Delta_{k+1} = \Delta_k,$$

where

$$G_j = \leftarrow M_1^\sim, \ldots, M_{i-1}^\sim, N_1^{j\sim}, \ldots, N_{q_j}^{j\sim}, M_{i+1}^\sim, \ldots M_n^\sim$$

if there is $L \leftarrow N_1^{j\sim} \ldots N_{q_j}^{j\sim} \in Q$ $(1 \leq j \leq m, m \neq 0)$.

(ff1-3) If L is not equal to any head of any clause by means of (ff1-2), then

$$H_{k+1} = H'_k, \Sigma_{k+1} = \Sigma_k, \Delta_{k+1} = \Delta_k.$$

(ff2) In case that $M_i^\sim = \sim L$ (a ground literal) and $L \in \Delta_k$,

$$H_{k+1} = H'_k \cup \{\leftarrow M_1^\sim, \ldots, M_{i-1}^\sim, M_{i+1}^\sim, \ldots M_n^\sim\},$$
$$\Sigma_{k+1} = \Sigma_k, \Delta_{k+1} = \Delta_k.$$

(ff3) In case that $M_i^\sim = \sim L$, $L \notin \Delta_k$, and there is a succeeding derivation $(\leftarrow L, \Sigma_k, \Delta_k) \Rightarrow_{suc} (\Box, \Sigma', \Delta')$,

$$H_{k+1} = H'_k, \Sigma_{k+1} = \Sigma', \Delta_{k+1} = \Delta'.$$

For the ground ELP, the presented procedure can be complete with respect to the model, which guarantees the soundness of the procedure. If the ground ELP is an infinite set, then the procedure is of infiniteness. From applicative views, the generated ELP to represent the objects in software engineering may be finite such that the procedure can work with respect to the model in the sense of soundness and completeness. For the soundness, see [32].

As regards the completeness of the contradiction-free procedure, the program with exceptions is to be assumed with its semantics. For the completeness, the similar technique as in [14, 30] may be available, although the technique is developed for the class of general logic programs containing only negation as failure but not explicit negation, and we have not completely proved that the technique is all right: The complete procedure for general logic programs may be developed to operate on the extended logic program with exceptions, by just replacing treatments of atoms with those of literals. A contradiction-free procedure to operate on the original program is regarded as behaving by simulating the complete procedure for the program with exceptions, by cutting off exceptions through the executions.

As another aspect, we have a problem of whether a non-grounded version of the coherence principle may be built-in or not, where a non-grounded version of negation as failure is in [24, 27, 28, 29]:

$\leftarrow L$ succeeds with the empty substitution
$\Rightarrow \leftarrow \neg L$ fails.

3. LOOP PARALLELIZATION WITH THE EXPLANATION PROCEDURE

In recent works, compilers often allow programmers to indicate parts that has no data dependence. The process of parallelization by such compilers is:

1. The compiler perform data dependence analysis.
2. The compiler transforms loops in which it knows there is no data dependence.
3. Programmers check the parallelized program and if there are loops that really has no data dependence but the compiler cannot find it, then programmers indicate it to the compiler.
4. The compiler transforms loops according to the indication of programmers.

The problem is an interface between programmers and the compiler: how programmers know the loops which really could be parallelized but are not by the compiler. The points are:

- There are three kinds of status about data dependence. The compiler knows that there are data dependences, the compiler finds that there is no data dependence, and the compiler cannot find that there is no data dependence.
- In general, there are many loops and many data dependences in a program. It is not easy for programmers to find a loop from the data of all of the loops.

We propose to use an extended logic program as an interface between programmers and a compiler. An extended logic program can naturally describe three kinds of status as positive literal, negative literal and negation as failure. An explanation reasoning procedure of an extended logic program is suitable for programmers to pick out a non-parallelized loop and check why the loop is not parallelized.

In general, a parallelizing compiler infers whether a loop can be executed in parallel or not:

- Check data dependences of the variables and arrays which are accessed more than once in the loop. This is an analysis of dependence vectors [5].
- If all the variables/arrays have no data dependence, the loop can be parallelized. Otherwise, it cannot.

These rules are represented as an extended logic program:

$$
\begin{array}{l}
parallelize_L \leftarrow \neg dependence_{V1}, \ldots, \neg dependence_{Vn} \\
\neg parallelize_L \leftarrow dependence_{V1} \\
\ldots \\
\neg parallelize_L \leftarrow dependence_{Vn} \\
\neg parallelize_L \leftarrow\sim \neg dependence_{V1} \\
\ldots \\
\neg parallelize_L \leftarrow\sim \neg dependence_{Vn}
\end{array}
$$

where $parallelize_L$ means the loop L can be parallelized, and $dependence_{Vi}$ means a variable or an array V_i in the loop has data dependence.

These rules are generated by the explanation reasoning procedure dynamically according to the information from the compiler. Also the declarations as below are added dynamically.

$$
\begin{array}{l}
\neg dependence_{V1} \leftarrow \\
..... \\
dependence_{Vi} \leftarrow
\end{array}
$$

Then the explanation reasoning procedure performs the abductive procedure on these rules. The naf set denotes the set of variables/arrays whose data dependences the compiler is uncertain about. If we give the information about the data dependence of them, the loop may be parallelized.

EXAMPLE 1. The following ELP, which is generated by the compiler, denotes the rules and declarations about a loop in Fig. 2.

$$
\begin{array}{l}
parallelize_L \leftarrow \neg dependence_a, \neg dependence_b, \neg dependence_c \\
\neg parallelize_L \leftarrow dependence_a \\
\neg parallelize_L \leftarrow dependence_b \\
\neg parallelize_L \leftarrow dependence_c \\
\neg parallelize_L \leftarrow\sim \neg dependence_a \\
\neg parallelize_L \leftarrow\sim \neg dependence_b \\
\neg parallelize_L \leftarrow\sim \neg dependence_c \\
\neg dependence_b \leftarrow \\
\neg dependence_c \leftarrow
\end{array}
$$

A compiler knows that the array b and c have no data dependence but it is uncertain about the array a, because of indirect references.

For a goal $\leftarrow \neg parallelize_L$, there are three rules whose head is $\neg parallelize_L$. If we choose

$$\neg parallelize_L \leftarrow\sim \neg dependence_b$$

the second goal is $\leftarrow\sim \neg dependence_b$. The failing derivation is invoked with $\neg dependence_b$, but there is a clause

$$\neg dependence_b \leftarrow$$

then $\leftarrow \neg dependence_b$ holds, $\leftarrow\sim \neg dependence_b$ does not hold, and $\leftarrow \neg parallelize_L$ does not hold.

If we choose

$$\neg parallelize_L \leftarrow\sim \neg dependence_a$$

the second goal is $\leftarrow\sim \neg dependence_a$. The failing derivation is invoked with $\leftarrow \neg dependence_a$. There is no rule whose head is $\neg dependence_a$, so $\leftarrow \neg dependence_a$ does not hold. Then $\sim \neg dependence_a$ holds, and $\leftarrow \neg parallelize_L$ holds. The naf set is $\{\neg dependence_a\}$.

It means the loop in Fig. 2 is not parallelized because of the uncertainness of data dependences of the array a. If programmers feel certain that there is no data dependence about the array a, they can inform the compiler of the information as adding a clause $\neg dependence_a \leftarrow$. Then the compiler can know that it can parallelize the loop.

```
      do 10 i = 1, 100
        b(i) = a(idx(i))
        c(i) = b(i) * a(idx(i))
        a(idx(i)) = a(idx(i)) * 2
10    continue
```

Figure 2: A loop with indirect references

```
      do 20 i = 3, 100          :1
      do 18 j = 5, 100          :2
        a(i,j) = b(i, j+5)      :3
        b(i,j) = a(i, j−1)      :4
        c(i,j) = a(i,j)         :5
18      continue                :6
20    continue                  :7
```

Figure 3: An example for the loop distribution

4. LOOP TRANSFORMATION WITH THE EXPLANATION PROCEDURE

We can know a certain loop transformation method is applicable or not to a loop, by checking dependence vectors of the loop. A dependence vector describes data dependence [4].

DEFINITION 1. A dependence vector $DepVec(S,T,v)$ on a variable v between a statement S and a statement T is a set of elements $dep(dv(S,T,v),k)$, where k is a sort of dependence: flow dependence, anti dependence or output dependence. $dv(S,T,v)$ is a vector which satisfy the following definition.

$$\begin{aligned} dv(S,T,v) &= (d_1,\cdots,d_n) \\ &= (d_{i1},\cdots,d_{in}) - (d_{j1},\cdots,d_{jn}), \end{aligned}$$

where $(d_{i1},\cdots,d_{in})$ and $(d_{i1},\cdots,d_{in})$ are the former and the latter iterations respectively for the execution of statements S and T.

We describe conditions of dependence vectors for applying a loop transformation method to a loop, as a logic program. By using an explanation reasoning, we can know that a certain loop transformation method can be applied to a loop or not, and parallelize the loop or a part of the loop.

Let us see an example. There is a loop transformation method called loop distribution which divides a loop into two loops. If it makes that one of the loops has no data dependence, then we can parallelize the loop. The loop distribution can be applied if there is no data dependence between the divided two loops.

The condition for applying a loop distribution can be described as a clause of a logic program as the following.

$$LoopDistribution(p,k) \leftarrow AllL(\sim ExistRevDep(S,T,p,k)),$$

where $ExistRevDep(S,T,p,k)$ is a literal which holds when there is at least one data dependence from statement T to statement S on the loop p when S is before the kth line of the source code and T is after. $AllL(f)$ is a literal which holds when f holds for any dependence vector in the specified loop.

```
# expara rules program.dvdata
Please input a literal:
LoopDistribution(i,4)
LoopDistribution(i,4) is TRUE. In the case
Success set is
        AllL:
        not ExistRevDep: S=3 T=4 #=a p=i k=5
        not ExistRevDep: S=3 T=5 #=a p=i k=5
        not ExistRevDep: S=3 T=4 #=b p=i k=5
Fail set is
        not AllL:
        ExistRevDep: S=3 T=4 #=a p=i k=4
        ExistRevDep: S=3 T=5 #=a p=i k=4
        ExistRevDep: S=3 T=4 #=b p=i k=4
```

Figure 4: The result for the query of the loop distribution

Fig. 3 shows two nested loops: the loop variable of the outer loop is i and the loop variable of the inner loop is j. We call the outer loop as $loop_i$ and the inner loop as $loop_j$. The $loop_i$ can be parallelized but the $loop_j$ cannot be parallelized because of the data dependences between the sentence 3 and 4 on the array a and b.

See the example to ask whether the loop distribution can be applied or not. Fig. 4 shows the output of our system when we input LoopDistribution(i,4) which is the query about the loop distribution of $loop_i$ between the sentence 4 and 5. The answer is TRUE that means we can divide the $loop_i$ into two parts: one is from the sentence 1 to 4, and the other is the sentence 5. Then we can parallelize the outer loop of the former part and the complete loops of the latter part.

5. CONCLUDING REMARKS

In this paper, we report that we apply the abductive procedure to loop transformation in a parallelizing compiler. We dynamically generate an extended logic program which includes rules and declarations according to the analyses by the compiler, and show why the compiler infers a loop cannot be parallelized.

In the field of parallelizing compilers, one of important topics is how we choose proper loop transformation methods. One of the solutions is that programmers may give additional information and assist the compiler [10, 23]. Our experience may be useful and improve the solution.

However, to apply our procedure to the real cases, more works will be necessary. We have to consider whether there is more suitable description of dependence vector to represent conditions for other loop transformation methods, whether the procedure on extended logic programs can be performed in reasonable time, how programmers pick up a loop from many loops in a program, and whether this method is useful in practical use.

6. REFERENCES

[1] J. J. Alferes, C. V. Damásio, and J. M. Pereira. A logic programming system for nonmonotonic reasoning. *J. Automated Reasoning*, 14:93–147, 1995.

[2] A. Analyti and S. Pramanik. Reliable semantics for extended logic programs with rule priotization. *J. of Logic and Computation*, 5:303–324, 1995.

[3] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *ACM Computing Survey*, 26(4):345–420, 1994.

[4] U. Banerjee. *Dependence Analysis for Supercomputing.* Kluwer Academic Publishers, 1988.

[5] U. Banerjee. *Loop Parallelization.* Kluwer Academic Publishers, 1994.

[6] K. L. Clark. Negation as failure. In *H. Gallaire and J. Minker (eds.), Logic and Databases*, pages 151–177, 1995.

[7] P. M. Dung. An argumentation-theoretic foundation for logic programming. *J. of Logic Programming*, 22:151–177, 1995.

[8] K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In *Proc. of 6th ICLP*, pages 234–255, 1989.

[9] M. Gelfond and V. Lifschitz. The stable model semantics for logic programs. In *Proc. of 5th ICLP*, pages 1070–1080, 1988.

[10] M. W. Hall, T. J. Harvey, K. Kennedy, N. McIntosh, K. S. McKinley, J. D. Oldham, M. H. Paleczny, and G. Roth. Experiences using the parascope editor: an interactive parallel programming tool. *SIGPLAN Notice*, 28(7):33–43, 1993.

[11] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. *J. of Logic and Computation*, 2:719–770, 1992.

[12] A. C. Kakas, R. A. Kowalski, and F. Toni. The role of abduction in logic programming. In *D. M. Gabbay, C. J. Hogger and J. A. Robinson (eds.), Handbook of Logic in Artificial Intelligence, Vol.5,*, pages 235–324. Oxford Science Publications, 1998.

[13] A. C. Kakas and P. Mancarella. Preferred extensions are partial stable models. *J. of Logic Programmimg*, 14:341–348, 1992.

[14] A. C. Kakas and F. Toni. Computing argumentation in logic programming. *J. of Logic and Computation*, 9:515–562, 1999.

[15] R. A. Kowalski and F. Sadri. Logic programs with exceptions. In *Proc. of 7th International Conference on Logic Programming*, pages 598–613, 1990.

[16] R. A. Kowalski and F. Toni. Abstract argumentation. *Artificial Intelligence Law*, 4:275–296, 1996.

[17] J. W. Lloyd. *Foundations of Logic Programming, 2nd, Extended Edition.* Springer-Verlag, 1993.

[18] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction removal within well-founded semantics. In *Proc. of 1st International Workshop on Logic Programming and Nonmonotonic Reasoning*, pages 105–119, 1991.

[19] P. M. Pereira, N. Joaquim, J. N. Aparício, and J. J. Alferes. Non-monotonic reasoning with logic programming. *J. of Logic Programming*, 17:227–263, 1993.

[20] T. Przymusinski. Every logic program has a natural stratification and an iterated least fixed point model. In *Proc. of 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 11–21, 1989.

[21] W. Pugh and D. Wonnacott. Eliminating false data depenedences using the omega test. In *Proceedings of the ACM SIGPLAN'92 conference on programming language design and implementation*, pages 140–151, 1992.

[22] M. Sasakura. Concentric circle diagrams for visualizing a reasoning process on an extended logic program. In *PDPTA'02, Volume I*, pages 253–259, 2002.

[23] M. Sasakura, K. Joe, Y. Kunieda, and K. Araki. Naraview: an interactive 3D visualization system for parallelization of programs. *International Journal of Parallel Programming*, 27(2):111–129, 1999.

[24] J. C. Shepherdson. Negation in logic programming. In *J. Minker (ed.), Foundations of Deductive Databases and Logic Programming*, pages 19–88, 1987.

[25] A. Van Gelder. The alternating fixpoint of logic programs with negation. *J. of Computer and System Sciences*, 47:185–221, 1993.

[26] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38:620–650, 1990.

[27] S. Yamasaki. A denotational semantics and dataflow construction for logic programs. *Theoretical Computer Science*, 124:71–91, 1994.

[28] S. Yamasaki. SLDNF resolution with non-safe rule and fixpoint semantics for general logic programs. *Theoretical Computer Science*, 160:283–303, 1996.

[29] S. Yamasaki and Y. Kurose. Soundness of abductive proof procedure with respect to constraint for non-ground abducibles. *Theoretical Computer Science*, 206:257–281, 1998.

[30] S. Yamasaki and Y. Kurose. A sound and complete procedure for a general logic program in non-floundering derivations with respect to the 3-valued stable model semantics. *Theoretical Computer Science*, 266:489–512, 2001.

[31] S. Yamasaki and M. Sasakura. Towards distributed programming systems with visualizations based on nonmonotonic reasoning. *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet(CD-ROM) 76*, 2001.

[32] S. Yamasaki and M. Sasakura. A contradiction-free procedure with visualization for extended logic programs. In *Proc. of SSGRR02, 4*, 2002.

[33] J.-H. You and L. Y. Yuan. On the equivalence of semantics for normal logic programs. *J. of Logic Programming*, 22:211–222, 1995.

Agent-Based Support for Requirements Elicitation

Chad Coulin, Didar Zowghi
Department of Software Engineering, University of Technology Sydney
PO Box 123 Broadway NSW 2007 Australia
{chadc, didar}@it.uts.edu.au

ABSTRACT

The elicitation of requirements is a difficult and expensive process but critical to the overall success of any system development. So far relatively little work has been devoted to providing intelligent tool support for this complex and labor-intensive activity. The quality of requirements from the elicitation process currently depends greatly on the experience and expertise of the participating requirements engineers, and the commitment and cooperation of the system stakeholders.

In this paper we describe an agent-based approach to intelligent tool support for requirements elicitation. Given the multiple roles a requirements engineer must perform during elicitation, we suggest a multi-agent system (MAS) may be developed as an intelligent assistant for this process. It is proposed that some of the tasks performed by requirements engineers during the elicitation process may be supported and in some cases automated by individual agents or several agents working cooperatively.

It is expected that the use of intelligent agents would produce better requirements in terms of their completeness, correctness, consistency and clarity. This would be achieved partly by improving the elicitation process through greater efficiency with respect to time and cost, and increased effectiveness by way of rigorous and structured execution.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *Elicitation methods, Methodologies, Tools.*

General Terms

Design, Theory.

Keywords

Requirements elicitation, intelligent tool support, agents, roles, multi-agent system.

1. INTRODUCTION

Requirements elicitation is a very complex process involving many activities with multiple techniques available to perform those activities [7]. The multi-disciplinary nature of tool support for requirements elicitation only adds to this complexity with strong relationships to the fields of knowledge engineering, artificial intelligence, information systems, cognitive psychology and the social sciences. This is in addition to the large body of work in the obvious and more general areas of systems, software and requirements engineering.

It is generally understood that requirements are elicited rather than captured or collected. This implies both a discovery and development element to the process [5]. Requirements may be elicited from a variety of sources including the many different types of possible stakeholders in the future system, and documentation and processes from the existing systems.

Few attempts have been made to develop intelligent tools to support requirements elicitation especially where there is direct interaction with human stakeholders without the need for a requirements engineer driving the process or the use of a semi-formal modeling and analysis technique. In order to improve the quality of requirements and the elicitation process itself we introduce the use of agents as intelligent support for the requirements engineer during this phase of system development.

The paper is structured as follows: Section 2 explains the meaning of agents as intelligent assistants in the context of the requirements elicitation process. Section 3 describes a multi-agent based approach to the requirements elicitation process. The use of agents for domain knowledge is examined in Section 4, and in Section 5 we investigate the particularly challenging area of discourse agents with respect to requirements elicitation. Finally in Section 6 we present a discussion with some conclusions and possibilities for future work.

2. INTELLIGENT AGENTS

We use the definition of an agent as a computer system situated in some environment that is capable of flexible autonomous action in order to meets its designed objectives [4]. In this definition there is additional importance placed on the term ‘flexible’ in that it refers to the responsive, pro-active and social nature of agents.

For our purposes an agent can further be described as an active software component or entity utilizing intelligent technology in terms of its communication with is environment and its dynamic behavior [9]. Another important point to make is that agents may be organized into a hierarchy or social structure in order to interact and perform tasks with other agents and entities within that system [2].

Therefore three key concepts concerning agents can be defined as autonomy, adaptation and cooperation [1]. By this it is meant that an agent should have the ability to make independent decisions without external intervention, be aware of its environment and able to make changes to its behavior accordingly, and interact with other agents and entities and in some cases work cooperatively with them [6].

WITSE '03, September 1, 2003, Helsinki, Finland.

Agents are different to objects in the traditional software engineering sense in that an agent must ultimately exhibit control over its own behavior [4]. However agents may depend on other entities such as knowledge bases and the Internet to be able to meet their goals and complete their tasks. We take the view that an agent may act as a proxy for a requirements engineer in order to perform one or more tasks on their behalf.

The advantages of using agents include the ability to manage and reference large amounts of real time and historical data, and have greater control and consistency over the output and results of the process. This includes assuring that equal attention is paid to both the problem and solution domains, and that the final documentation and presentation conforms to accepted standards.

Agents could also enable the automation of some of the more mundane tasks a requirements engineer is required to perform during the elicitation process. This is particularly important given the time consuming nature of requirements elicitation especially when dealing with complex systems and large organizations.

3. A MULTI-AGENT APPROACH

3.1 Roles of the Requirements Engineer

Multi-agent systems are typically very complex and difficult to develop. Despite this it is more likely that a multi-agent system would be developed to support some of the many activities performed by a requirements engineer as opposed to a single 'fat' intelligent agent providing all the functionality. This is mainly due to the nature and behavior of agents, and the state of technology currently available.

We can begin to investigate the implementation of a multi-agent agent system for requirements elicitation by examining the various roles performed by the requirements engineer during this process.

A fundamental part of the entire requirements engineering process is related to project management. This activity involves more than the obvious decision-making and prioritization tasks. Project managers are also commonly required to initiate meetings with stakeholders, produce project status reports to inform stakeholders of progress, remind stakeholders of their responsibilities, and answer questions from stakeholders regarding the project, the process, and the system being developed.

When eliciting requirements by conducting interviews the requirements engineer does not only ask questions and record the responses but also must guide and assist the participants in answering these questions in order to elicit the most correct, complete and relevant information. The interviewer is also responsible for ensuring that participants feel comfortable and confident with the process in order to achieve the best possible results from this activity.

Conflicts between stakeholder requirements are inevitable. When this occurs the requirements engineer is often required to act as a mediator in working towards a suitable resolution. All elicited requirements must be validated and verified against each other and the previously established goals of the system. This may involve various semi-formal and formal modeling and analysis activities.

Requirements engineers are often required to assume the roles of the developer community during requirements elicitation such as system architects, designers, programmers and testers. Decisions made during the requirements stage will inevitably effect the later phases of system development.

The responsibility for the output of the elicitation process also lies with the requirements engineer. Typically this exists in the form of a requirements document or detailed system model. This role is particularly important as it represents the results of the elicitation process and forms the foundation for the subsequent project phases. Evaluation of the elicitation process and the work performed by the requirements engineer is based on these resultant artifacts which in some situations will form the basis of a contractual agreement as in the case where a system is to be developed for an organization by an external supplier.

3.2 Elicitation Activity Agents

For each of the roles detailed in the previous subsection we can identify one or more possible types of intelligent agent application to support the required activities.

3.2.1 Personal Assistant Agents

Personal assistant agents could be responsible for proactively organizing and driving the project tasks of human stakeholders and assisting the execution of these tasks by providing the necessary guidance proactively and when requested.

3.2.2 Information Acquisition Agents

Information acquisition agents could be used to search through existing documentation and knowledge bases to validate existing requirements and to discover new ones.

3.2.3 Elicitation Technique Agents

Elicitation technique agents could be used to interview, model, analyze and document requirements based on stakeholder input. Additional sub-agents could be developed to check for conflicts and consistency of requirements during elicitation, inform the stakeholders of the specifics, and advise on possible solutions.

Surveys are one of the most common elicitation techniques used by requirements engineers and include questionnaires and structured and unstructured interviews [3]. Agents may be used to design and propose questions based on meta-models and schemas, case stories, and abstractions of previously developed systems. It is also possible to conduct surveys based on goal refinement and other knowledge acquisition techniques.

Scenario and task analysis involves 'walking' stakeholders through existing or proposed system operations and defining each possible step and exception condition. An agent may be used to model, simulate and incorporate feedback for the described operations from either a user or system perspective. The use of graphical representation is especially useful during this process and could be incorporated into the behavior of the agent.

Some elicitation techniques are inherently more suited to the possibility of implementation through agents than others. These elicitation agents may be arranged into a structure with a parent or decision agent responsible for selecting the appropriate elicitation technique and agent to employ depending on information about

the system stakeholders and other environmental constraints such as time and the availability of resources.

Agents may also be used to support rapid prototyping and other agile methods used for requirements elicitation. This could involve the use of interactive software construction agents during the early stages of system development. Further investigation is also required into how agents may be used to support observational and ethnographical techniques during requirements elicitation.

In practice requirements elicitation is an iterative process and typically a combination of techniques is used to discover and develop system requirements. For example a requirements engineer may conduct a follow-up discussion with the system stakeholders after validating the information gathered in a prior interview by observing the existing system in use.

3.2.4 Administration Management Agents

Administration management agents could perform much of the day-to-day administration tasks often assigned to the requirements engineer such as reserving meeting rooms and other resources, producing and distributing regular status reports, and reminding stakeholders of their obligations.

3.3 Agent Coordination

In the multi-agent system required to perform the various tasks a requirements engineer must conduct during the requirements elicitation process the agents would need to run concurrently and not interfere or conflict with each other. In some cases it may be necessary for agents to work cooperatively in order to satisfy their individual goals. For this reason it is critical to ensure that agents in this type of environment do no have conflicting goals and adhere to defined coordination strategies and protocols.

We can take the simple example of a decision agent required to determine which elicitation techniques are best utilized for a particular project and which of the stakeholders to involve. In this case the requirements engineer informs the web-based agent of basic project and organizational details. The agent has access to a knowledge base of various elicitation techniques, the conditions under which they can be performed, and their respective strengths and weaknesses.

This is combined with information from stakeholders captured via an online questionnaire on their individual details, the goals and constraints of the project, the problem domain, and the availability of resources. The system includes a feedback mechanism also via an online questionnaire to evaluate the relative success or failure of the selected elicitation techniques. As a result the agent is able to reference past experiences for future questionnaires and decisions, and therefore continuously improving the performance of the system.

Furthermore a series of sub or contractor agents may be employed to perform the necessary subsequent tasks. An administration agent could be used to arrange a meeting where the appropriate elicitation agent would develop and conduct a structured interview based on the predetermined high-level goals and constraints defined for that particular system development project. The parent decision agent would need to communicate the number of participants and the type of meeting to the administration agent which may then in turn schedule the meeting based on the electronic calendars of individual participants and reserve an appropriately sized and equipped conference room.

A personal assistant agent would be responsible for reminding the stakeholder participants of the interview and determining any prerequisite tasks each stakeholder must perform prior to the meeting. The elicitation agent may decide to use a template, model or analogy to support the interview or employ an information acquisition agent to retrieve documentation of the existing system to use as the basis for further inquiry. Here the elicitation agent would need to communicate to the information acquisition agent the scope of the search and the type of information required.

It is beyond the scope of this paper to delve further into the specific details of the intelligent and behavioral aspects of the proposed agents although it is tempting to do so. However we can see from this example how the communication and coordination between agents in a multi-agent system is paramount to its success.

4. AGENTS FOR THE DOMAIN

Domain knowledge represents an important part of requirements engineering however the collection of domain knowledge is a very time consuming process. During requirements elicitation both the problem and solution domains need to be examined. This type of information can be exploited for requirement engineering in a variety of ways.

Requirements engineers will use previous experience in the domain as a kind of mental template for group discussions and interviews. Domain analogies and abstractions of existing situations are used as baselines to acquire information in order to identify and model possible solution systems. This also provides the opportunity to reuse specifications from like and unlike domains, and validate new ones against existing domain knowledge as detailed in the work of Sutcliffe and Maiden [8].

Agents could be implemented as experts to provide assistance in not only the collection of domain knowledge but also its presentation for further information acquisition activities. We can define a domain expert as having an extensive knowledge of the domain area, the ability to identify similarities and differences between domain instances, and access to a catalogue of existing examples in the given domain for reference.

Therefore it is possible to conceptualize an intelligent agent working in cooperation with a domain knowledge base to support the requirements engineer during the elicitation and modeling of requirements.

5. DISCOURSE AGENTS

Discourse agents that can actively participate in conversations directly with one or more human subjects at a time present us with some unique and complex challenges. Requirements engineers often use group discussions involving multiple stakeholders in elicitation. Examples of this include focus groups and joint application development (JAD) groups. In these cases the requirements engineer may or may not be required to have significant domain expertise in order to facilitate discussions.

We can investigate the difficulties of developing discourse agents by examining the role of group discussion facilitator that is often

performed by the requirements engineer during elicitation. In this environment the requirements engineer is not only responsible for managing and guiding the other participants on the topics established for discussion but also to inquire when more information or clarification is required on a subject and maintain the relevance of all discussions to the problem at hand. It is important that the requirements engineer ensures that all parties represented in the discussion group are given sufficient opportunity to voice their positions and provide appropriate input to the conversations.

The requirements engineer is sometimes also required to negotiate on behalf of absent stakeholders, validate previously established requirements with the current group of participating stakeholders, and mediate disagreements between stakeholders.

To perform these tasks effectively a requirements engineer must not only be able to just see and hear the participants but also analyze their body language and speech patterns. The requirements engineer uses this type of information throughout the elicitation process to gauge the level of importance and understanding of requirements within the group and detect possible conflicts and concerns the stakeholders might have.

It is difficult to envisage an intelligent agent capable of conducting this role given the technologies currently available. In this case an extremely complex system of both software and hardware would be required with substantial multimedia and sensory capabilities. Despite this is it interesting to look at how intelligent agent technologies might contribute to this area in future research.

6. DISCUSSION

From our preliminary investigation it has been determined that there are many conceivable applications of agents in the activities performed during requirements elicitation however whether or not one or more of these can be successfully implemented into an intelligent multi-agent system to support the requirements engineer in this process is yet to be completely examined both theoretically and practically.

Multi-agent systems provide us with some significant new opportunities and advantages over the more traditional expert systems. Typically multi-agent systems integrate several general tasks that can be personalized to the individual user as opposed to expert systems that perform a limited number of specific tasks in a fixed manner for all users. Agent-based applications are inherently more active, adaptive and mobile than expert systems in their behavior and interaction with users and other information sources.

It is worth mentioning that such multi-agent systems as proposed in this paper are currently very expensive and complicated to design and build. Additional obstacles such as the cultural change and acceptance of multi-agent systems within organizations would also need to be addressed for this type of technology to be adopted. Given this it is more realistic that single agents will be developed for generic applications such as searches and scheduling and then modified and applied to requirements engineering as opposed to multi-agent systems being designed and built specifically for requirements elicitation activities.

More research on agent development, architecture and application is still needed, especially in social environments and multi-agent systems where the issues of conflicts between agents, load balancing, belief revision, and the re-organization of agent commitments are still being examined.

It is not suggested that agents present a substitute for the roles of the requirements engineer but instead that certain elements of the requirements elicitation process may be automated, supported and improved through the implementation of an intelligent agent-based system.

We believe that the development of information systems presents a particularly good opportunity to apply this type of technology in both the early and later stages of requirements elicitation due to the large body of work in this area and the volume of available and relevant literature, domain expertise and case stories.

7. REFERENCES

[1] Dardenne, A., van Lamsweerde, A., and Fickas, S. 1993. Goal-directed requirements acquisition. Science of Computer Programming 20, 3-50.

[2] Giorgini, P., Kolp, M., and Mylopoulos, J. Multi-Agent and Software Architectures: A Comparative Case Study in Proceedings of AAMAS ’02, Bologna, Italy, May 2002.

[3] Goguen, J. A., and Linde, C. Techniques for Requirements Elicitation in Proceedings of RE ’93, San Diego, CA, January 1993, 152-164.

[4] Jennings, N. R., Sycara, K., and Wooldridge, M. 1998. A Roadmap of Agent Research and Development. Autonomous Agents and Multi-Agent Systems 1, 7-38.

[5] Jirotka, M., and Goguen, J. (Ed.) Requirements Engineering: Social and Technical Issues. Academic Press, London, UK, 1994.

[6] Kim, M., Lee, S., Park, I., Kim, J., and Park, S. Agent-Oriented Software Modeling in Proceedings of APSEC ’99, Takamatsu, Japan, December 1999, 318-325.

[7] Nuseibah, B., and Easterbrook, S. Requirements Engineering: A Roadmap in Proceedings of The Future of Software Engineering, Limerick, Ireland, May 2000, 35-46.

[8] Sutcliffe, A., and Maiden, N. 1998. The Domain Theory for Requirements Engineering. IEEE Transactions on Software Engineering 24 (3), 174-196.

[9] Zhou, Y., and Pan, Y. 2000. Agent-Oriented Analysis and Modeling. ACM SIGSOFT Software Engineering Notes 25 (3), 36-40.

DCBL: A framework for Dynamic Control of Behavior based on Learning

Patrice VIENNE
INSA Lyon, PRISMa, Bat. B. Pascal,
69621 Villeurbanne Cedex, France
pvienne@if.insa-lyon.fr

Jean-Louis SOURROUILLE
INSA Lyon, PRISMa, Bat. B. Pascal,
69621 Villeurbanne Cedex, France
sou@if.insa-lyon.fr

ABSTRACT

This paper describes the DCBL (Dynamic Control of Behavior based on Learning) framework that aims to aid developing systems that dynamically control application behavior. It relies on learning abilities to make the application able to improve autonomously its behavior even when the context changes. The framework is used to increase the QoS of a set of applications. The behavior is controlled to reduce the risk that the supplied QoS exceeds a specified minimal level. The DCBL framework significantly decreases software development cost while the learned behavior still fulfills the application requirements.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features—*Frameworks*; I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Reinforcement Learning, Quality of Service, Software development

1. INTRODUCTION

Within a computer system, applications running in a changing context may have several alternatives to fulfill their objectives. A decision making process is then needed to provide the application with suitable behavior. Such applications have to dynamically adapt their behavior to improve it according to some criteria. In this paper, the criterion is an utility indicator that measures the overall system Quality of Service (QoS: "A set of qualities related to the collective behavior of one or more objects", ISO[2]). Then the behavior of the system applications is adjusted in order to maximize this utility. The more direct way to handle this problem is to make each application able to adapt its behavior based on its own utility indicator. However, a specific solution has to be developed for each application and interactions between applications are not taken into account (i.e., the overall utility cannot be maximized). Thus, this solution is not reusable and has a very high development cost. A better option is to extract from the application all the elements contributing to the decision making process and to gather them in a control system middleware layer. Now, the overall system utility is easier to maximize since a global view is available. Moreover, only one decision process have to be built and well defined interfaces with applications makes it reusable.

To deal with behavior adaptation, two ways are possible. The first way is to find a specific algorithm. Unfortunately, this is a tremendous work as every possible change in the context must be envisaged during the development. Furthermore, the general decision problem is NP-difficult[3], and therefore a heuristic should be used[1]. The second way lies on learning techniques. They cannot ensure that the maximum utility will be reached, but as the learning system gains experience it progressively increases the relevance of the control. Furthermore, the learning performs autonomously (i.e. without human intervention)[4] and online (i.e. applications run simultaneously). As the context may change, learning is performed almost uninterruptedly, making the system able to adapt its behavior at any time. The use of such learning techniques presents two major advantages: (i) the control system is very general and widely reusable; (ii) developers do no more need to find detailed solutions for the system behavior.

This paper describes DCBL, a framework for developing a control system using a learning technique. DCBL also provides the system with a mechanism that aims to finely control the trials and errors inherent in the selected learning algorithm. First the used learning technique is outlined, and then the control system architecture and its working principles are detailed.

2. LEARNING TECHNIQUE

Learning techniques can be classified into three main groups: supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, the expected output y_i is available for each possible input x_i. Learning requires searching for the function f by successive modifications such that: $\forall i, f(x_i) = y_i$. The evaluation of the function f is based on the prediction error $f(x_i) - y_i$. In most cases, learning is performed on a training set and then the discovered function is used in a real context.

Unlike supervised ones, unsupervised learning techniques do not require knowing outputs. Parameters are adjusted only

based on input data and a set of predefined constraints. These learning techniques are usually used for clustering and classification problems.
Reinforcement learning [6] is a mix of supervised and unsupervised techniques. Unlike supervised learning, the expected output does not need to be available for a given input. However, a reinforcement signal characterizing the behavior is required. The learning process relies on this signal that takes the form of penalties and rewards.
This last group of learning techniques seems the most appropriate to solve the problem of adaptation in a changing context: (i) expected outputs are usually unknown, and they may change in dynamic contexts; (ii) it is often easy to find information about behavior quality.

2.1 Q-Learning

Q-Learning [7] is a reinforcement learning technique that copes with state succession. Let $\mathcal{A}$ denote the set of available actions and $\mathcal{S}$ the set of available states. The goal of Q-Learning is to estimate the value $Q(s,a)$ for each pair $< s,a >$ where $s \in \mathcal{S}$ and $a \in \mathcal{A}$. $Q(s,a)$ stands for the long-run payoffs obtained by taking action a in the state s. In other words, $Q(s,a)$ is the benefit to take action a in state s.
For each time step, the state s_{t+1} follows the state s_t by taking the action a_t. While doing this, the reinforcement signal r is received and $Q(s_t,a_t)$ is updated:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\Big(r - Q(s_t,a_t) + \gamma \max_{a\in\mathcal{A}}\big(Q(s_{t+1},a)\big)\Big) \quad (1)$$

The correction of $Q(s_t,a_t)$ is reduced using a learning rate coefficient α in the range $[0,1]$ and can be split into two parts: $r - Q(s_t,a_t)$ makes the estimated value of the benefit tends towards the value of the received reinforcement r, and $\gamma \max_{a\in\mathcal{A}}\big(Q(s_{t+1},a)\big)$ stands for the benefit of the new state, characterized by the best state-action pair reduced by the coefficient γ. In practice, the learning algorithm endlessly repeats the following steps: (1) receiving the reinforcement signal and the new state, (2) updating the benefit of taking the selected action in the previous state using the received reinforcement signal and the current state, (3) choosing and executing an action.

2.2 Action selection

Action selection is an important step in the learning process and several strategies are possible. The first way consists in always choosing the action a given the state s so that $Q(s,a)$ has the highest value. This strategy, called greedy action selection, cannot generally be used to find optimal behavior: early convergence makes the learning process inefficient. A subset of actions is used regardless of the others. On the opposite, a uniformly random action selection is a good choice to visit all state-action pairs, but the resulting behavior has no relationship with the learned one.
The way to select the action should be a judicious mix of exploration (random action selection) and exploitation (greedy action selection). Furthermore, it is preferable to select more frequently an action that leads to a high-valued benefit, and less often an action that leads to a low-valued one. The "softmax" function based on Boltzmann formula [5] uses these principles and give the probability to select the action a given a state s:

$$\mathbb{P}(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{b\in\mathcal{A}} e^{\frac{Q(s,b)}{\tau}}} \quad (2)$$

τ is a parameter used to give more or less importance to an action that leads to a high-valued benefit. The more τ has a high value, the more the action selection tends to be done randomly. On the opposite, when the value of τ is close to 0, action selection looks like the greedy one.

3. DCBL FRAMEWORK

The behavior of a system can be characterized by its overall utility. It stands for the quality of the service that the system provides from the user's point of view. The Dynamic Control of Behavior based on Learning (DCBL) framework copes with execution context adaptation of applications based on a reinforcement learning technique. It aims to increase the probability to maintain system utility above a predefined threshold.

3.1 General view

A major advantage of DCBL is that elements contributing to the decision making process are extracted from the applications and arranged to form the control system. In their remaining parts, applications only deal with execution of the choices made in the control system. Both parts of the system interact via input and output interfaces (fig. 1).
The control system lies on a reinforcement learning algorithm, which requires the current state and the reinforcement signal: (i) the state is deduced from the perceptions supplied by applications (i.e., a set of values that specify the situation of the system in its context); (ii) from the monitor values provided by the application, the system utility is built and its variation gives the required reinforcement signal. A synthesis step realizes the transformation from perceptions and monitors to state and utility.
The decision system being separated from applications, it can handle several applications simultaneously. Only the simple case of one application is presented in this document, but all principles remain valid in the general case.

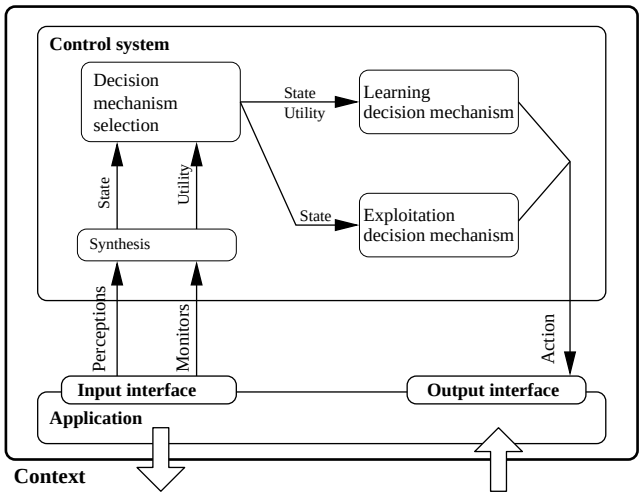

Figure 1: DCBL framework

3.2 Operating principles

3.2.1 Utility and minimal utility

System utility is a control system built-in data. Its estimation depends on the developer's description of the monitors: a valid area of value is provided for each of them. The utility is maximal if all monitor values are in the center of their validity area. The utility tends to the minimal utility if at least the value of one monitor tends to the limit of its validity area.
The control system aims to discover a behavior that maximizes the system utility and reduces the risk that the utility goes below the minimal utility.
This description of the system room for maneuver based on each monitor makes the developer's work much more easy as he only has to handle one dimension of the problem (i.e. one monitor) at a time.

3.2.2 Decision mechanism

The system decisions are made using a Q-Learning algorithm associated with an action selection strategy based on "softmax" function. This requires state-action pairs exploration that may lead to a system utility below the *minimal utility*. Moreover, the system learns in an incremental way depending on how the system interacts with its context: which event occurs and when... Therefore, utility variation with time is not a strictly increasing function although it tends to increase. In other words, the system may adopt a behavior that is unable to maintain the utility above the minimal threshold. To avoid such a situation, the control system includes an additional decision mechanism. This exploitation mechanism uses a greedy action selection without learning. It should be initialized in such a way that the system is able to stay in its room for maneuver.
A decision mechanism selection step is responsible for the choice between learning and exploitation mechanisms.

3.2.3 Operating scenario

The activity of the decision system is organized in time steps. At each step a complete decision making process takes place: synthesis of inputs, selection of a decision mechanism, action selection with the decision mechanism and application of the selected action. On figure (2), several time steps show in a simplified way how the decision mechanism is selected. The space of states is composed of four utility areas separated by thresholds. At time step t, the system is in state s_t and the utility is *good*. The learning mechanism is used in order to discover a new behavior that fits better with the context. At time step $t+3$, the utility becomes *insufficient*, meaning that learning may well increase the risk to lead the utility below the minimal threshold. The learning decision mechanism is then replaced by the exploitation one. Exploitation takes place (from $t+3$ to $t+5$) until the system utility reaches *good utility* threshold at time step $t+6$. Then the system starts a new learning period.

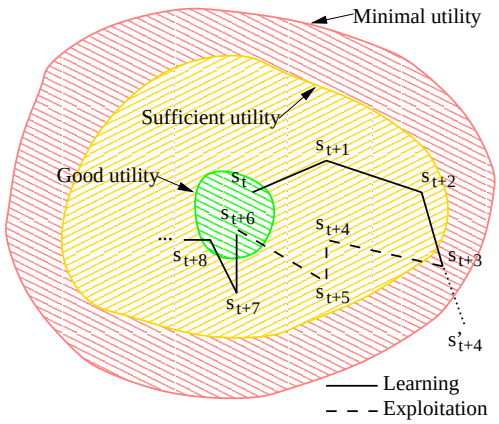

Figure 2: Simplified decision mechanism selection scenario

3.2.4 Utility threshold determination

During the scenario of figure (2), the only use of the learning mechanism may lead to state s'_{t+4}, in which bounds are exceeded. To reduce the risk to reach such a configuration, DCBL relies on the well-tried behavior of the exploitation decision mechanism. As *sufficient* and *good utility* thresholds are used to change the decision mechanism, they are essential in the utility control process.
The *sufficient utility* threshold depends on the ability of the exploitation mechanism to control the utility of the system. During execution, this threshold is adjusted to reduce the risk of constraint violation. The *good utility* threshold depends only on exploitation mechanism: the value should be low enough to be reached at exploitation time, and high enough to increase the *sufficient utility* area necessary for learning. This threshold is also adjusted dynamically.

3.3 Operating modes

Exploitation mechanism relies on its associated behavior to maintain the system QoS within a specified range. If a better behavior is discovered using learning abilities, the exploitation mechanism should be updated to include the improvements. With such an upgrade ability, the system constantly adapts itself to its changing context. Exploitation mechanism modification changes the benefit of taking an action rather than others in a given state.
Upgrading the system is not risk-less: the behavior may be upgraded although it does not improve the system QoS. To avoid these side effects, three operating modes are introduced: (i) *Exploitation mode* simply uses the exploitation mechanism; (ii) *Learning mode* uses the learning mechanism without any restriction; (iii) *Evaluation mode* is a restricted use of the learning mechanism: the "softmax" action selection is replaced by the greedy one, and learning does not take place anymore. This operating mode evaluates the relevance of the discovered behavior to decide whether the exploitation decision mechanism should be upgraded or not.
Q-Learning representation makes partial upgrade possible (i.e., state-action pairs can be modified independently): after evaluation, states will be upgraded when encountered enough times without significant decrease in utility. Partial upgrade is useful to reduce evaluation duration with little effects on the control system reliability.

3.4 Indicators and operating mode selection

At each time step, the system tries to change its operating mode. Mode change is based on indicators that represent the internal configuration of the system.
Two utility indicators are introduced to characterize the behavior associated with each decision mechanism. They are

meant to be the relevance of each decision mechanism. Utility indicators estimation is based on a recent average of the global utility while using the specified decision mechanism. These indicators are named *U_exploitation* and *U_learning*. To be relevant, utility indicators must be evaluated over enough consecutive time steps. Moreover, as the context may change in time, the reliability of indicators decreases if they are not updated in the current operation mode. To address this problem, reliability indicators are associated with each operating mode (*R_exploitation*, *R_learning* and *R_evaluation*): their value increases when the system is in the corresponding operating mode, and decreases otherwise. Reliability thresholds are used to decide whether these indicators are *good*, *sufficient* or *insufficient*, just as utility thresholds for system utility measurement (cf. 3.2.4).
Finally the failure indicator depends on the ability of the exploitation mechanism to maintain the system in its room for maneuver. The system fails if the utility goes below the *minimal utility* threshold and it comes back to non-failure if its utility reaches the *good* threshold (fig. 2).
On figure (3), the operating mode selection is detailed depending on the current mode of the system. At *Upgrade* stage, exploitation behavior is modified to take into account the discovered improvements.

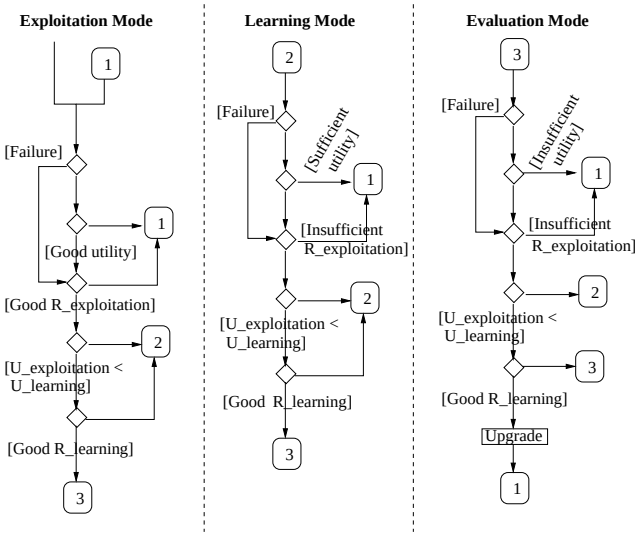

Figure 3: Operating mode selection detail

3.5 Parameter variation

Dynamically adjusting some parameters improves the system QoS. Two main principles are described.
First, the system room for maneuver can be reduced if a better behavior is discovered during execution. With a smaller room for maneuver, the system average QoS will be significantly improved. All the system parameters are updated to fit the new local room for maneuver.
Second, some parameters depend on the relationship between system and time steps succession. For example, how many time a newly discovered behavior should be evaluated before exploitation *upgrade* ? Those parameters can be dynamically adjusted to take into account the time steps succession, making DCBL more efficient and application-independent.

4. CONCLUSION

DCBL is a framework for Dynamic Control of Behavior based on Learning. As a major advantage, DCBL maintains the system QoS above a minimal threshold. Trial and errors learning requires the exploration of a subset of the space of states to discover new behavior that better fits the context. Several introduced mechanisms increase the system average QoS, while maintaining the current one above a threshold. Another advantage of DCBL is its ability of self adaptation to a changing context. A behavior that was efficient at a time may progressively be replaced by another one that turns out to be better in the current context. To keep self adaptation ability, the system does not stop learning, even when a very efficient behavior has been found. However, the system learns less to take advantage of the discovered behavior.
Using DCBL does not require to perfectly know the application area. Only objectives and an initial, relevant enough, behavior need to be defined by the developer. Then the system will autonomously evolve upon this ground to improve the delivered QoS. Moreover, a good way to use DCBL is to develop a unique application for several slightly different contexts. At run time, each application will evolve to find a behavior that fits well its specific context.
As the control system does not have specific relationships with the operating system, it will be easily made reusable. On the other hand, applications do not include the decision making process, which is common to all applications.
This work only begins and all the possibilities have not been studied yet. The use of the control system can be extended to any system lying on a decision making process. Furthermore, it would be interesting to include in DCBL other learning techniques.

5. REFERENCES

[1] J. L. Contreras and J. L. Sourrouille. A Framework for QoS Management. In *TOOLS'39*, pages 183–193. IEEE Press, 2001.

[2] ISO/IEC. Quality of Service, Guide to Methods and Mechanisms. Technical Report ISO/IEC 13243 Draft 1.0, ISO/IEC, 1997.

[3] C. Lee and D. Siewiorek. An Approach for Quality of Service Management. Technical Report CMU-CS-98-165, CMU, 1998.

[4] L. Steels. When are robots intelligent autonomous agents ? *Journal of Robotics and Autonomous Systems*, 15:3–9, 1995.

[5] P. Stefán and L. Monostori. On the Relationship between Learning Capability and the Boltzmann-Formula. In *Engineering of Intelligent Systems*, pages 227–236. IEA/AIE, 2001.

[6] R. S. Sutton and A. G. Barto. *Reinforcement Learning : an Introduction*. MIT Press, 1998.

[7] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.