Representing and Reasoning about Context in a Mobile Environment

Marius Mikalsen[1] and Anders Kofod-Petersen[2]

[1] SINTEF Telecom and Informatics,
7465 Trondheim, Norway
mariusm@sintef.no
[2] Department of Computer and Information Science,
Norwegian University of Science and Technology,
7491 Trondheim, Norway
anderpe@idi.ntnu.no

Abstract. This paper describes an approach to representing contextual information through a domain independent solution. It, further more, shows how case-based reasoning can be used to reason about user-situations.
The solutions described is implemented within a context-aware system supporting users in an mobile environment.

1 Introduction

The users of mobile computers today are bringing an ever increasing amount of computational power and storage along. Most mobile users today are also equipped to access the Internet via a wide array of different carriers. With this movement of the computer from the desktop to the ubiquitous paradigm described by Weiser [1], the computer system should now adapt to the user's situation, instead of the user adapting to the computer.

Situation adaption, or services and products customisation/personalisation, consists of two major components. Some type of *contextual information* is required, such as information about the user, the user's environment, etc. However this is not sufficient; some mechanism to *reason* about this information is also required. When these two components are present, a *context-aware* system will be able to deliver *context-sensitive* services to the users.

Even though a lot of research has been conducted within context-aware systems, the core term *context* is not yet a well defined concept. As a consequence of this the interesting issue of knowledge use in context-aware systems is almost invisible. Most of the research until now have been focused on the technical issues associated with context, and the syntactic relationships between different contextual concepts.

This paper describes an approach to automatic situation assessment in a mobile environment within the AmbieSense research project. Even though the AmbieSense system includes both mobile and fixed parts, this paper focuses on the mobile part of the system. The three major issues covered here are: the open-ended context model, the multi-agent system, and the reasoning mechanism.

2 Related Work

The research field of context-awareness has seen a lot of research covering many diverse topics. A lot of this research is relevant to the research presented here. However, given the limited scope of this paper, we have decided to focus on two projects relevant to our research: the *Context Toolkit* by Dey [2], and the *Reflective Middleware Solution* by Capra et. al. [3].

The Context Toolkit [2] aims to add the use of context to existing non-context-aware applications and to evolve existing context-aware applications.

The Context Toolkit introduce the context widget, which is responsible of acquiring a certain type of context information, and make this information available to applications. Applications access the widget by using poll and subscribe methods. Context widgets operate independently from applications that use them. Context widgets make the distribution of context sensing devices in the architecture invisible to the context-aware applications, mediating all communication between applications and components.

The context interpreter incorporates interpretation functionality to try to predict future actions or intentions of users. The interpreter accepts one or more contexts and produces a single piece of context. One example may be to get all contexts from all widgets in a conference room, and determine that a meeting is occurring. This functionality requires the programmer to write the actual code that performs the interpretation for this specific problem.

The Context Toolkit delivers a standardised way of implementing the syntactical part of context-aware systems. However, the reasoning about contextual information must be implemented for each domain and application. Thus, making it flexible only in the design and implementation phase, and not in run-time.

Another approach to aggregate context has been developed by by Capra et. al. [3]. In this system a marriage of reflection and metadata is suggested as a means to create a middleware that give applications dynamic access to information about their execution context.

In this view, middleware is seen as a network middleware. Network middlewares sit on top of a network operating system and provides application developers with higher levels of abstractions, hiding complexities introduced e.g. by distribution (e.g. disconnections).

Network middlewares have been designed and work successfully on stationary computers, but they appear not to be suitable for the mobile setting of the following reasons: Firstly, the interaction primitives (e.g. distributed transactions) assume high-bandwidth connection of components, as well as constant availability. This is not the case in mobile systems, where unreachability and low bandwidth is the norm rather than an exception. Secondly, completely hiding implementation detail from applications makes little sense. Mobile systems need to detect and adapt to drastic change occurring in the environment, such as battery power. If we have complete transparency, the middleware need to make decisions on behalf of the application. Applications however, make more efficient and better quality decisions based on application specific information.

Reflection and meta data are used to build the system that support context aware applications. Applications pass metadata to the middleware. This metadata constitutes a policy as to how the applications want the middleware to behave as a result of a specific

context occurrence. As context and application needs changes continuously, one cannot assume that meta data are static, therefore applications use reflection mechanisms offered by the middleware to inspect their own meta data, and possibly alter it according to changing needs. The meta data is standardised using XML Schemas.

3 AmbieSense

AmbieSense is a project in the Information Society Technologies (IST) Programme of the European Union. The goal is to develop a set of software and hardware tools to facilitate context aware computing. The AmbieSense framework can be used to build ubiquitous information channels in the surroundings, capable of delivering the right information to the right time to the mobile traveller.

3.1 System overview

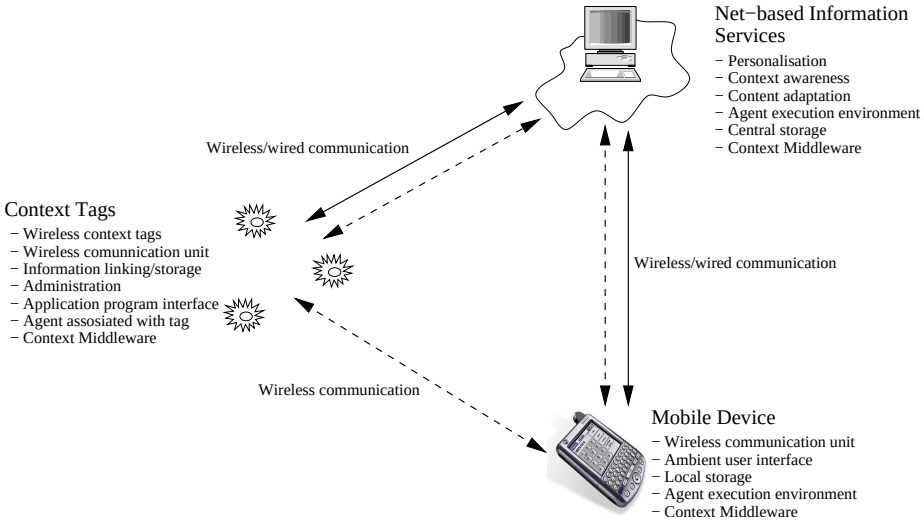

Fig. 1. Overall Architecture

The mobile system is one compound in an architecture that also contains "Context Tags" (Bluetooth beacons) primarily used for communications and location, and Net-Based Information Services (see Figure: 1). Since this work is concerned with contextual understanding on mobile devices, the overall architecture will only be touched briefly. Interested parties are directed to the projects website (www.AmbieSense.com) or a more thorough description of the architecture in Myrhaug and Göker [4].

One of the Context Tag's primary assignments is to supply the location. It can also distribute localised information, such as the menu of the particular restaurant, where it is installed. The more advanced Context Tags can offer both local services or a connection to services located on the net.

The Net-Based Information Services are divided into two major categories: *i*) The AmbieSense connected services that offer personalised information services tailored to

the AmbieSense system. *ii*) The more general information services that are available on the Internet.

The mobile system is divided into three major parts: the Context Middleware (Section: 3.3), the multi-agent system (Section: 3.4), and the reasoning mechanism (Section 3.5).

3.2 Context

Given the domain independent nature of the context middleware, a broad and non-excluding definition of context is needed. We extend the definition of context given by Dey [7], applying the following definition to context:

> Context is the set of suitable environmental states and settings concerning a user, which are relevant for a situation sensitive application in the process of adapting the services and information offered to a user.

We believe that this is a pragmatic definition of context that allows application developers to efficiently rule out information that is not context in their particular application domain. At design time, developers can ask the question; is this information relevant for adapting our services and information? If the answer is no, the information is discarded as not being context, and excluded from the context model. This flexibility leads to an open context model that only defines the taxonomic structure is in the design phase (see Figure 2).

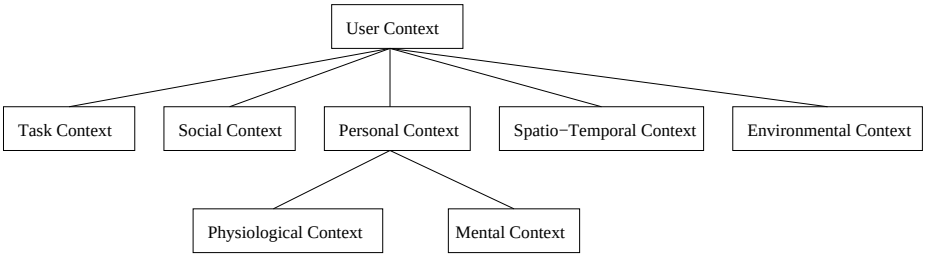

Fig. 2. User Context in the AmbieSense project

As part of the AmbieSense framework, the context is divided into five main categories (a more thorough discussion can be found in [8]): *i*) Environmental context: This part captures the users surroundings, such as things, services, light, people, and information accessed by the user. *ii*) Personal context: This part describes the mental and physical information about the user, such as mood, expertise, disabilities and weight. *iii*) Social context: This describes the social aspects of the user, such as information about friends, relatives and colleagues. *iv*) Task context, the task context describe what the user is doing, it can describe the user's goals, tasks, activities, etc. *v*) Spatio-temporal context: This type of context is concerned with attributes like: time, location and movement. The different aspects of the contexts are attribute-value tuples that are associated with the appropriate contexts.

This work postulates that there exist a goal or task in any situation. It would be futile to identify a situation unless there is some task connected to it - no matter how

mundane. This is most obvious when dealing with users, where a situation implies that there is a problem that needs to be solved; such as the possible situation “hungry user”, which implies the goal of *not hungry user*, leading to the task *provide food*, with a subtask ‘*locate food*.

3.3 Context Middleware

The utility of context aware services has already been demonstrated by [5]. The problem today is that many context aware applications custom build proprietary infrastructures for their context management [6].

The main contribution of the context middleware is to provide a generic and user-friendly context management infrastructure, by collecting and maintaining context. The context middleware allow applications to utilise relevant context information, paying little or no attention to the details of context management.

The context information available in the context middleware is provided from some context sources. Examples of sources are mobile users (through a user interface), applications, software agents and sensors. Clients may be other context sensitive applications or other software agents. The sources and clients interact with the context middleware, using services relevant to their particular needs, without integrating with each other directly.

The context middleware offers the following relevant features: *Context representation and validation*: The context middleware supports the creation of valid context representations compliant to a given context template. *Context storage and retrieval*: The users current context can be stored so it can be utilised in the future. In the context middleware contexts are stored in a context space, as context history, a current context, or a future context. *Publish and subscribe context*: context aware clients use subscription mechanisms of the context middleware to indicate that they are interested in notifications when context changes as a results of sources publishing context.

Context Templates As the name implies, the context template is a template for context representations. The purpose of the context template is to provide a pattern that all contexts that is added to a context space, as compliant to this template, must be valid to.

As we have seen, a generic context service such as the context middleware must make few assumptions about context needs across different domains to ensure usability. But, given the uniqueness of various domains, application developers will need a mechanism to constrain context relevant to their domain. The context template enables developers to constrain context and define contextual information valid within their particular domain. This is a common rationale for any other domain model. Using the context template one can define context structure, and attributes and valid values. Using the validation mechanism of the context middleware, one can ensure that a given context instance is valid towards the context template for a particular domain.

Context instance *A* would be allowed into the context space, while context instance *B* would be discarded as not being a valid context within this particular domain.

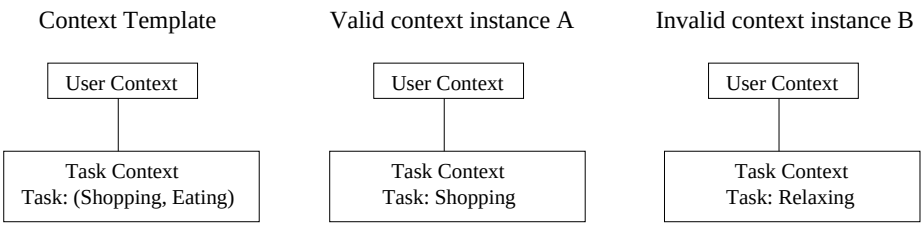

Fig. 3. Example context template and context instances (valid (A) and invalid (B))

Context Space The context middleware implements a context space [9]. The context space abstraction is essential to capture the difference between transient and persistent context [6].

Transient context reflects the environment at a single point in time, whereas persistent context represents a recurrent pattern of transient context. The fact that Hans Inge eats a Danish pastry at 10:30 AM in the morning is transient context, and the fact that he eats a Danish every morning at 10:30 is persistent context. An application (or agent) may deduce persistent context from retrieving context from history, and the persistent context it predicts, it can store this in the context future (context cache). The current context is the set of contexts and attributes currently relevant. When not relevant any longer, it can be stored into context history.

3.4 Agents

As a part on the AmbieSense system, the mobile device holds the agency that identifies user situations and solves the problems associated with the current situation.

Agents have been chosen for various reasons. One of the most important is the modularity that an agency supports. This flexibility is a great benefit when new application agents are implemented into the system.

The agency consists of three kinds of agents:

- The framework agents supplied by the agent platform Jade [10]
- The core agents handling the situation detection, and goal decomposition, which in turn are connected to the context middleware.
- The application agents, each handling one specific application, such as map construction or news gathering.

The context agent communicates with the context middleware, that maintains the space of contextual data. This agent receives new *current contexts* from the context middleware, translates them to Jade ACL messages, and sends them to the relevant receivers. The case-based reasoning agent utilises the Creek [11] CBR system to identify the current situation. Based on this identification, the agent notifies the facilitator about this particular situation. This notification is the set consisting of the situation with the corresponding contextual information, and the goal associated with it. Once the facilitator has reached it's goal, the solution will be returned to the context agent.

The facilitator uses the Unified Problem-solving Method description Language (UPML) [12] to maintain an overview of the application agents' services and for handling

the task decomposition. It will receive the situation and task description from the context agent, decompose the task into the different sub-tasks required, and recruit the correct application agents for solving the tasks.

Application agents are responsible for solving their own particular tasks. At present four different application agents exists:

- Map agent, who can access the map server and supply map suited to the particular context information.
- News agent, who can gather relevant news.
- Information agent, who can gather information that is generally available on the net.
- Airport agent, who can solve airport-related problems.

When new application agents are to be introduced into the system, the programmer needs only to know the context ontology, and how to specify the capabilities of this agent in UPML, and the agent is basically ready to be a part of the system.

3.5 Reasoning

In most of the research in context-aware systems, the problem of filtering the vast amount of contextual information that is available, in such a way that the identification of important constellations of the contextual information is feasible, has not been thoroughly addressed. Case Based Reasoning is a promising method for this.

Case Based Reasoning [13] is concerned with adapting to new situations by remembering similar earlier experienced situations (cases). CBR has historically been used in large monolithic systems. This work applies CBR as a lightweight reasoning mechanism that is capable of running on a small mobile device.

The reasoning mechanism is split into two different parts. The on-line part that resides on the user's mobile device, and the off-line reasoning that resides on the user's backbone system.

On-line reasoning Different types of contextual information can arrive in a very diffuse fashion, e.g. time is continually flowing into the system, whereas location might be pseudo static. Since CBR works on discreet cases, the continuous values flowing into the system must be made discreet. The context agent receives a *Current Context* from the Context Middleware, translates it to fit the ontology used in Jade, and sends it to the CBR agent.

Once a new context arrives, the CBR cycle is activated (see Figure 4). The system will try to retrieve a known context or case, and classify the current situation based on the retrieved one. When the situation has been classified, the associated goal is presented to the task decomposition agent. After the agency has handled the problem the case will be stored in the case base in a triplet consisting of: *i*) the contextual information describing the situation, *ii*) the problem associated with the situation, and *iii*) the solution constructed by the application agents.

Since the user is expected to experience at least a few different situations a day, the storage of the cases will quickly fill up the mobile device. This potential vast amount

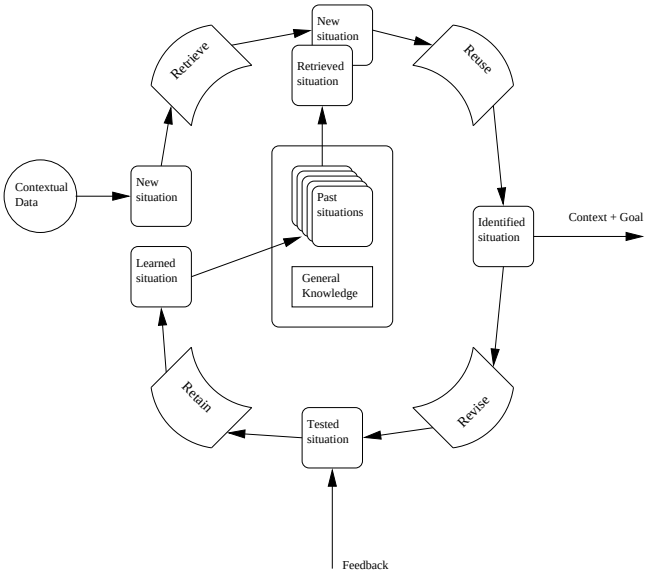

Fig. 4. CBR Cycle

of cases will also severely hamper the searching process for the CBR mechanism. To remedy this, some of the reasoning process is moved into the user's backbone servers.

Off-line reasoning There are potentially two main problems with the use of Case-Based Reasoning for identifying situations: the storage problem, and the problem of indexing and searching.

First and foremost is the problem of storing the, potentially vast, amount of cases constructed during run time. To solve this, the user will have personal persistence storage available on the user's home network. This storage will be used for storing the cases, and will be synchronised when the user has an up-link. This large amount of data does not only affect the amount of storage space needed, it will also severely affect the indexing and matching algorithm used.

To remedy this a generalisation process will occur on the home net. Similar cases will be grouped into prototypical cases, e.g. everything that the 600 business meetings has in common will constitute the prototypical business meeting situation. These prototypical cases will be part of the on-line case base, and be used in the every day reasoning process. Generalisation of cases is a well known research area within CBR, for an overview see for an example [15]. This case base is structured by prototypes, which are generated on the basis of the amount of similar parameters in the point-cases.

4 A Hungry User in an Airport – An Example

In this scenario we will show how the system can detect that a user is hungry in OSL Airport Gardermoen and assist him in finding food to his liking.

The user enters the restaurant area of OSL Gardermoen, where a context tag is mounted. From the current user context (see Figure 5) on the client, we know that the user prefers Italian food. The time attribute in spatio-temporal context shows that the time is a quarter past one. Other parts of the personal context shows that it has been more than five hours since he last used his credit card. As the user enters the context tag's Bluetooth zone, the context information on the tag is transferred to the client. This information is merged with the current user context, and represents a new context. New information in the current user context is now that the location in Spatio-temporal context is *Eating area*.

The context agent receives a new current context from the context middleware. The context agent translates the context instance to a Jade ontology instance, which is sent to all interested parties, including the CBR agent. The CBR system can now take this situation case and try to find a matching case in it's case base. A hungry user case is matched against existing cases, based on the information presented above. The matching case is a case that identifies a task get-food (see Figure: 6), and can leave it to the facilitator to accomplish it.

The facilitator knows what agents are available on the mobile system, and what services they offer. The facilitator can furthermore look up the services that are available in the new current context, under Service Providers in the Environment Context. This information is then used to decompose the task of `get-food` into `find-restaurants` and `match-restaurants-to-preferences`.

Once the solution has been returned to the Context Agent, it can remind the user that it's time for lunch, by telling him that: "there is a nice Italian restaurant three minutes away. The restaurants web-page claims that the service is acceptable, and that you can get a meal for only 11 €".

```
Case 0 (Current Context)
   TASK: Identify User Need
   TASK STATE: In Process
   SOCIAL CONTEXT: Departing
   PERSONAL CONTEXT:
     PREFERENCE:
       Italian Food
       Guardian Newspaper
     FINANCIAL:
       Last-CC:
         08:04:00
         [Joe's Breakfast Club]
   SPACIO-TEMPORAL:
     Current Location :
       OSL - Eating Area
     Current Time: 13:15:00
   ENVIRONMENTAL:
     Entity: OSL-CT-TAG-12
     Services:
       find-restaurants@OSL-CT-TAG-12
```

Fig. 5 Current Context

```
Case 1 (Hungry User Situation)
   TASK: Get Food
   TASK STATE: Accomplished
   GOAL: Not Hungry
   SOCIAL CONTEXT: Departing, Hungry
   PERSONAL CONTEXT:
     PREFERENCE:
       Italian Food
       Le Figaro Newspaper
     FINANCIAL:
       Last-CC:
         diff(Current Time. Last-CC) >= 5 h)
   SPACIO-TEMPORAL:
      Current Location : Eating Area
      Current Time: range[11:00 - 14:00]
   ENVIRONMENTAL:
```

Fig. 6 Hungry User Case

5 Conclusion and further work

The work presented here describes an approach to handle two important issues in context-aware applications; namely the possibility of specifying behaviour based on

context in the implementation phase, and the problem of aggregating contextual data from many and diverse sources.

The context middleware allows for specification of context templates at implementation time, where the programmer can define behaviour for a particular situation. Further more, the context middleware handles the aggregation of contextual information from many sources and presents it in a standardised manner to the reasoning part of the system.

As stated earlier it is not sufficient to gather and aggregate contextual data, some kind of reasoning about this data is required. This work proposes the use of case-based reasoning as a method for context-sensitive applications. As this is ongoing research, experimental results are still to come. However, work by Zimmermann [14] suggests that CBR is a promising method for identifying the correct combination of contextual information that leads to a good situation understanding.

An indoor user test at Oslo Airport was carried out in the beginning of May this year. The test aimed at showing how the aggregation and merging of contextual information was perceived by users in a real environment. The analysis of the test data was at the time of writing not yet finished.

6 Acknowledgements

This work is part of the AmbieSense project, which is supported by the EU commission (IST-2001- 34244).

References

[1] Weiser, M.: Some computer science issues in ubiquitous computing. Communications of the ACM **36** (1993)

[2] Dey, A.K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology (2000)

[3] Capra, L., Emmerich, W., Mascolo, C.: Reflective middleware solutions for context-aware applications. In: Lecture Notes in Computer Science. (2001)

[4] Myrhaug, H.I., Göker, A.: Ambiesense – interactive information channels in the surroundings of the mobile user. In Stephanidis, C., ed.: Univeral Access in HCI, 10^{th} International Conference on Human-Computer Interaction. Volume 4., Lawrence Erlbaum Associates (2003) 1158–1162

[5] Bristow, H.W., Baber, C., Cross, J., Wooley, S.: Evaluating contextual information for wearable computing. In: Proceedings of the Sixth International Symposium on Wearable Computers, IEEE Computer Society (2002)

[6] Lei, H., Sow, D.M., Davis, J.S.I., Banavar, G., Ebling, M.R.: The design and applications of a context service. Mobile Computing and Communications Review **6** (2002) 44–55 ACM SIGMOBILE.

[7] Dey, A.K.: Understanding and using context. Personal and Ubiquitous Computing **5** (2001) 4–7

[8] Göker, A., Myrhaug, H.I.: User context and personalisation. In: Workshop proceedings for the 6^{th} European Conference on Case Based Reasoning. (2002)

[9] Myrhaug, H.I.: Towards life long and personal context spaces. In: Workshop on User Modelling for Context-Aware Applications. (2001) Available at: http://orgwis.gmd.de/ gross/um2001ws/papers/myrhaug.pdf.
[10] Bellifemine, F., Poggi, A., Rimassa, G.: Jade - a fi pa-compliant agent framework. Technical report, Centro Studi e Laboratori Telecomunicazioni (1999) Part of this report has been also published in Proceedings of PAAM'99, London, April 1999, pagg.97-108.
[11] Aamodt, A.: A knowledge-intensive, integrated approach to problem solving and sustained learning. PhD thesis, University of Trondheim, Norwegian Institute of Technology, Department of Computer Science (1991) University Microfi lms PUB 92-08460.
[12] Fensel, D., Motta, E., van Harmelen, F., Benjamins, V.R., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., Musen, M.A., Plaza, E., nd R. Studer, G.S., Wielinga, R.: The unifi ed problem-solving method development language upml. Knowledge and Information Systems Journal (KAIS **5** (2003)
[13] Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications **7** (1994) 39–59
[14] Zimmermann, A.: Context-awareness in user modelling: Requirements analysis for a case-based reasoning application. In Ashley, K.D., Bridge, D.G., eds.: ICCBR 2003, Case-Based Reasoning Research and Development. LNAI 2689, Springer-Verlag (2003) 718–732
[15] Maximini, K., Maximini, R., Bergmann, R.: An investigation of generalized cases. In Ashley, K.D., Bridge, D.G., eds.: ICCBR 2003, Case-Based Reasoning Research and Development. LNAI 2689, Springer-Verlag (2003) 261–276