Project Acronym: GiraffPlus
Project Title: Combining social interaction and long term monitoring for promoting independent living
Grant agreement no.: 288173
Starting date: 1st January 2012
Ending date: 31st December 2014

D 3.2 Integrated and refined reasoning infrastructure and deployed prototypes

WP related to the Deliverable:	WP3 Intelligent Monitoring and Adaptation Services
Nature:	Prototype
Dissemination Level :	Public
Version:	V1
Author(s):	Lars Karsson, Lia Silva-Lopez, Jonas Ullberg
Project Participant(s) Contributing:	Lars Karsson, Lia Silva-Lopez, Jonas Ullberg, Federico Pecora
Contractual Date of Delivery:	2013-07-01
Actual Date of Delivery:	2013-07-01

Document History

Version	Date	Type of editing	Editorial
V1.0	2013-07-01	First complete version	ORU

Deliverable Summary

In this report we present the results after the first year and a half in WP3 regarding Intelligent Monitoring and Adaptation Services, which consists of two major components: context recognition and configuration planning. The prototypes of these two components have been further developed, and their integration is more advanced than after the first year. The context recognition can now automatically infer what state variables are needed to monitor specific activities. The configuration planner no longer expects to receive all goals (state variable requests) at the same time but can instead extend an existing configuration in order to accommodate new goals.

Contents

1 Introduction

In this document we present the progress made between M12 and M18 in WP3 regarding Intelligent Monitoring and Adaptation Services, which consists of two major components: context recognition and configuration planning. In the previous report of WP3, Deliverable 3.1 at M12, we presented prototypes of these two components, as well as a simple integration. We also presented a demo in which the GiraffPlus system at the Ängen test site was automatically configured and activities were inferred and displayed. In this report, we present a more advanced integration of the two components in which the context recognition, given a set of requested activities, can infer what state variables to observe in the environment, and in which the configuration planner can respond to a stream of requests from the context recognition and modify the current configuration accordingly. We also present work on developing heuristics for the configuration planner.

Beside these improvements, we have made a considerable effort to update the interfaces to the evolving GiraffPlus system, including the new middleware and the new structure of the database, as well as to the Data Visualization, Personalization and Interaction Services. Some of the former work is reported in D2.2, and some of the latter work will be reported in D3.1.

From a software perspective, the integration between the two components is also changing: the configuration planner was intially a completely standalone component but is now becoming an integrated part of the data preprocessing module of the context recognition. The preprocessing module was previously set up manually (see D3.1.), and contained what from the configuration planner's perspective could be seen as processing functionalities. With the new integration this is done automatically by the configuration planner. The latter will also in time have available a larger library of preprocessing tools (functionalities).

2 Relevant WP3 tasks and deliverables

The following tasks end deliverqbles in WP3 have been active or become active soon:

- Task 3.1 Long-term context recognition (OrU)
- Task 3.2 Configuration planning for efficient activity recognition (OrU)
- Task 3.3 Integration of context inference and configuration planning (OrU)
- Task 3.4 Long-term trend analysis (OrU), starting M18.
- Task 3.5 Deployment (OrU, CNR-ISTC, MDH)
- D3.1) Context inference and configuration planning prototypes: Initial prototypes of context recognition and configuration planning systems. A report will describe the prototypes. [month 12]
- D3.2) Integrated and refined reasoning infrastructure and deployed prototypes: A prototype of an integration of context inference and configuration systems, in which the former can give tasks to the latter. A report will describe the prototype. [month 18]

3 Context recognition: Inferring state variables from activities

Recall that in order to infer context, the generated set of intervals are combined with a model consisting of rules like the following:

$$\textbf{Cooking}\ \texttt{Equals}\ \textbf{Stove.on} \wedge \textbf{Cooking}\ \texttt{During}\ \textbf{Kitchen.presence}.$$

Here, **Cooking** is an activity, which can be inferred from the state variables **Stove.on** and **Kitchen.presence**. The state variables shoulds be properties and relations that can be observed through sensors.

In the previous deliverable (D3.1), there was a fixed set of activities that were monitored all the time. In addition, what state variables to monitor was manually specified. When the GiraffPlus system was started, all these variables were immediately sent to the configuration planner, which then generated a configuration.

For this deliverable, the context recognition can be given activity requests at different points in time. For each activity request, the context recognition goes through the rules with the activity on the left side and forms a set of the state variables on the right side. The latter are then sent to the configuration planner. As there might be activities defined in terms of other activities (and not state variables) in a hierarchical fashion, this process might involve several levels of rules.

The requests for activities are later supposed to come from the Data Visualization, Personalization and Interaction Services (DVPIS), where secondary users select activites of interest. Eventually, the secondary users will also use the DVPIS to select or modify the inference rules of the context recognition.

4 Configuration planning: extending configuration with new goals

Recall that a configuration is a set of selected sensors, actuators and computational processes, collectively called functionalities, that exchange information in order to solve a task. This task may be to observe one or more state variables, and/or to change some state variables through actuation. Some tasks may also require several steps, in particular if actuation is involved.

A configuration planning problem consists of a goal, an initial state and a set of available functionality instances. The goal is given in the form of a set of information and/or causal requirements that need to be satisfied. The planner then converts these elements into a partial configuration that contains a stub functionality instance with the same causal and information requirements as the goal contains. The initial state is also transformed into a functionality, with the state variables of the initial state as effects.

The planner works by identifying unsatisfied inputs and preconditions, collectively called flaws, in the current partial configuration, selecting one of those flaws and extending the partial configuration in different ways in order to satisfy that input/precondition. The planner then continues to work recursively on the extended configurations. When there are no flaw left, an admissible configuration has been found.

In order to handle a stream of goals, the configuration planner has been extended as follows:

- Once an admissible configuration has been found, the planner remembers this configuration.
- When additional goals are received, a new goal functionality is added to the current configuration.
- The planner is restarted, identifies the flaws in the new goal functionality and (if possible) generates a new admissible configuration.

5 Configuration planning: heuristics

In order to have a high-performance configuration planner, we have developed a number of heuristics inspired by histogram-based techniques used by constraint satisfaction algorithms. These heuristics are of two types:

1. Heuristics for ordering configurations: the search front of the planner consists of a set of partial configurations (search nodes), and these heuristics are intended to guide search towards the most promising partial configurations in the search front.

2. Heuristics for selecting the next requirement to satisfy in a given partial configuration (search node): there are typically a number of open requirements in a partial configuration, and these heuristics are for determining which one of those to focus on first.

The following heuristics utilizes a histogram built with the frequencies of the sources in every avaialbe functionality instance. Sources are associated in bins that share the same property and format. A second histogram is built with the frequencies of the requirements in a similar fashion.

5.1 Heuristics for ordering configurations

We have selected a number of heuristics that can be used to prune the search front according to different criteria and sorting it. Each heuristic is intended to use local information.

5.1.1 No support pruning (NSP)

This heuristic is useful to prune away configurations in which any requirement cannot be satisfied by any other source from the functionality instances in the current configuration (unsupported requirement). The heuristic value becomes minus one (-1) when a configuration contains at least one unsupported unsatisfied requirement, and one (1) when the configuration contains no unsupported or unsatisfied links.

5.1.2 Sum variance ratio (SVR)

The Sum variance ratio (SVR) heuristic aims to favor smaller values of variance in the histogram of sources, as well as shallower configurations with less requirements to satisfy is the smallest value in the source histogram, while also favoring configurations with better support for their unsatisfied links (the total number of sources for all requirements).

5.1.3 Highest smallest support (HSS)

The highest smallest support (HSS) heuristic aims to favor configurations with the highest smallest support, while favoring smaller configurations and available requirements. This heuristic is only reliable when chained with a heuristic that previously pruned away nodes with functionalities with no requirements.

5.2 Heuristics for selecting the next requirement to satisfy

For choosing the next requirement to satisfy, we consider the following heuristics:

5.2.1 Fail-first (FF)

This heuristic is based on the Fail-First principle by Haralick and Elliott Haralick and Elliott [1980]. It selects the most failure-prone unsatisfied requirement in a partial configuration, by sorting a histogram combining causal and information sources in increasing order of frequencies, and selecting the first element.

5.2.2 Rho heuristic

The Rho heuristic, which is a heuristic for constrainedness, was proposed by Gent et al. [1996]. Rho branches into the children that maximize the solution density of a problem by choosing the requirement with most and/or the tightest constraints. Rho favors expanding over the requirement that has the smallest *fraction* of sources that can satisfy it. We can calculate such fraction by dividing the number of sources that can satisfy the requirement, by the number of available requirements. We can also use the number of available requirements to discriminate over information and causal sources, or we could consider them overall, with different results.

References

Ian Gent, Ewan MacIntyre, Patrick Presser, Barbara Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Principles and Practice of Constraint Programming CP96*, pages 179–193. Springer, 1996.

Robert M Haralick and Gordon L Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.