Collaborative Development within a Social Robotic, Multi-Disciplinary Effort: the CARESSES case study

Roberto Menicatti*[†], Carmine T. Recchiuto*[†], Barbara Bruno[†], Renato Zaccaria[†] Ali Abdul Khaliq[‡], Uwe Köckemann[‡], Federico Pecora[‡], Alessandro Saffiotti[‡], Ha-Duong Bui[§], Nak Young Chong[§], Yuto Lim[§], Van Cu Pham[§], Nguyen Tan Viet Tuyen[§], Nicholas Melo[‖], Jaeryoung Lee[‖], Maxime Busy[¶], Edouard Lagrue[¶], Jean–Marc Montanier[¶], Amit Kumar Pandey[¶] and Antonio Sgorbissa[†]
Corresponding author's email: `carmine.recchiuto@dibris.unige.it`

Abstract— **In many cases, complex multidisciplinary research projects may show a lack of coordinated development and integration, and a big effort is often required in the final phase of the projects in order to merge software developed by heterogeneous research groups. This is particularly true in advanced robotic projects: the objective here is to deliver a system that integrates all the hardware and software components, is capable of autonomous behaviour, and needs to be deployed in real-world scenarios toward providing an impact on future research and, ultimately, on society. On the other hand, in recent years there has been a growing interest for techniques related to software integration, but these have been mostly applied to the IT commercial domain.**

This paper presents the work performed in the context of the project CARESSES, a multidisciplinary research project focusing on socially assistive robotics that involves 9 partners from the EU and Japan. Given the complexity of the project, a huge importance has been placed on software integration, task planning and architecture definition since the first stages of the work: to this aim, some of the practices commonly used in the commercial domain for software integration, such as merging software from the early stage, have been applied. As a case study, the document describes the steps which have been followed in the first year of the project discussing strengths and weaknesses of this approach.

I. INTRODUCTION

It is commonly agreed that the coordination and integration within complex research projects involving software development is an open issue, in particular when project partners are distributed across different time zones, cultures and languages.

This is extremely relevant in the industrial - commercial domain, where deadlines and timing constraints related to the release of new features and products should be respected as much as possible, and where the quality of the produced software is of the utmost importance. As a consequence, in recent years, a number of practices have been taken into account as basic principles for software development, with the aim of reducing the time needed for the integration of software written by multiple developers, without compromising its quality: Continuous Integration, Continuous Delivery and Continuous Deployment are some of these practices, which share the common view that integrating source code as frequently as possible, reliably releasing products to customers and using tools for software control can guarantee the overall optimization of time and resources [1], [2].

Far from being similar to commercial activities, scientific research projects share with them some characteristics that make software integration critical: they usually foresee a work plan spanning a number of years, they involve developers from different research entities, different countries and with different backgrounds, and their output should be a system including software components reliable enough to be made available to other researchers external to the project, and hopefully start a process of industrial exploitation.

However, joint scientific projects may lack a central coordination unit for software development and, as a result, research partners tend to develop and test their own software during the whole project, with limited communication channels, and a big effort is then required for the integration activities at the very end of the project[1]. In recent years, the adoption of industry-strength practices for software integration and development in the context of scientific research activities turned out to be successful, greatly easing the management and coordination of complex collaborative projects (valid examples are the projects DREAM [3] and BRICS [4]).

Given these premises, it is evident that scientific research could greatly benefit from the application of some of the practices usually applied in the commercial domain for software integration. The article presents the case study of CARESSES, a multidisciplinary research project on socially assistive robotics, started in January 2017 and involving 9 partners from the EU and Japan, in which the issue of software integration has been taken into account from the early stage (CARESSES stands for Culturally-Aware Robots and Environmental Sensor Systems for Elderly Support, `http://caressesrobot.org`). The article is structured as fol-

*R. Menicatti and C. Recchiuto equally contributed to this work.
[†]DIBRIS, University of Genova, Genova, Italy
[‡]AASS, Örebro University, Örebro, Sweden
[§]JAIST, Japan Advanced Institute of Science and Technology, Japan
[‖]Dept of Robotic Science and Technology, Chubu University, Japan
[¶]Softbank Robotics Europe, Paris, France

[1]These problems have been recently faced in the ERF2018 workshop "Packaging, Releasing and Maintaining software for robots", moderated by Andriy Petlovanyy

lows: Section II presents the project at a glance, underlying the most critical aspects related to integration. In section III, the steps for the collaborative development of the software and all integration activities conducted in the first year of the project are reported. Finally, Section IV presents conclusions.

II. The CARESSES project

The CARESSES project (Fig. 1) is a 37-months multidisciplinary project aimed at building culturally competent robots, i.e. robots able to match the culture, customs and etiquette of the person that they are assisting, while autonomously reconfiguring their way of acting and speaking [5]. The project starts from the observation that, while cultural competence has been deeply investigated in Nursing Literature (in particular, in the so-called Transcultural Nursing research field [6]), it has been only marginally introduced in the robotics domain, even in projects dealing with assistive robots [7]. Thus, the project is based on the idea that assistive robots that are more sensitive to the user's cultural identity could have a great impact on older persons' quality of life, reducing a caregiver's burden and improving efficiency and efficacy. To this aim the project involves six European and three Japanese partners: the presence of a multidisciplinary consortium, involving partners with a background in robotics, AI, Human-Robot Interaction, Transcultural Nursing, Social Psychology, Evaluation of complex public health interventions, and professional Health-Care makes particularly complex communication between partners and - consequently - the integration of research outcomes into a common system to be deployed and tested.

Fig. 1. Logo of the Caresses project

The project foresees a testing phase[2] where the assistive robot will directly interact with a number of older persons and with their informal caregivers. In order to prove the project assumption, the robot will be endowed with knowledge about the cultural group the person belongs to and specific knowledge about her (Cultural Knowledge Base), planning abilities for autonomously choosing the more appropriate action to execute (Cultural Sensitive Planner and Execution Module), and a wide range of sensorimotor capabilities for implementing plans and interacting with the person and with sensors in the environment, possibly updating the knowledge base whenever new information with cultural relevance is acquired (Culture-Aware Human-Robot Interaction Module). It is evident that all these aspects are strictly interconnected to each other, and therefore a high level of software integration is required.

[2]Testing will be led by University of Bedfordshire, UK

For this reason, in the context of CARESSES, much attention has been paid to the integration of the software modules developed by different partners, since the early stages of the project. During the first year, developers located in three different European countries (University of Genova, Italy; Örebro University, Sweden; SoftBank Robotics, France) and in Japan (JAIST)[3] have used off-the-shelf tools for collaborating on shared tasks, adapting them to the peculiar context of a social robotics research project.

III. Collaborative Development Steps

A. Definition of Software Components and Functional Architecture

Within the first two months of CARESSES (M1-M2), the global functional architecture of the system, with the analysis of the principal software modules and the message exchanged between them, has been defined. The functional architecture has been built starting from scenarios prepared by CARESSES Transcultural Nursing experts, psychologists and Health-Care professionals[4], describing daily routines of older persons and the kind of assistance they may receive from a robot companion[5].

As mentioned in Section II, the functional software architecture of CARESSES is based on three modules: the *Cultural Knowledge Base* (CKB), the *Cultural Sensitive Planner and Execution Module* (CSPEM) and the *Culture-Aware Human-Robot Interaction Module* (CAHRIM).

1) CKB: It contains a-priori general knowledge about each cultural group and specific knowledge about the user, which is built partly at set-up and partly through the interaction with the person (inputs from CAHRIM). Knowledge is stored in the form of an ontology (a formal naming and definition of the types, properties and interrelationships of the entities that exist for a particular domain of discourse [8]), complemented with probabilistic information in the form of a Bayesian Network (to merge generic cultural knowledge at a national level with individual preferences, thus avoiding stereotyped representations). Thanks to the information stored in the ontology, CKB provides CSPEM with information about the available (and preferred) planning and action operators, goals, norms, habits, beliefs, values, etc., each possibly depending on the person's cultural group as well as on her individual preferences.

2) CSPEM: It computes the plan for the robot to reach its goals and sends the request for actuating the plan to CAHRIM, based on the cultural information received by CKB and additional messages provided by CAHRIM (i.e. the current state of the robot, the user and the environment, as well as user requests).

[3]CARESSES includes another robotic partner, i.e., Chubu University, Japan. The integration of software components provided by this fifth partner starts in the second year of the project

[4]Middlesex University, UK; Nagoya University, Japan; Advinia Health-Care, UK

[5]`http://caressesrobot.org/en/2018/03/08/caresses-scenarios-and-guidelines-available`

3) CAHRIM: It is responsible for the interaction between the user and the robot and controls the robot and the smart environment. It receives from CSPEM requests to execute verbal or sensorimotor actions, and it returns information (possibly related to cultural factors) to update CKB about the person's goals, habits, beliefs, values, and preferences. From a functional point of view CAHRIM can be seen as split in different software blocks, including the management of the robot and of the smart environment.

Figure 2 shows the three modules of the CARESSES architecture, and the messages exchanged between them. Different CARESSES partners are responsible for the development of the three components above, thus making things more complex: University of Genova for CKB, Örebro University for CSPEM, JAIST, SoftBank Robotics, Chubu University for CAHRIM.

B. Selection of the middleware

The next step (months M3-M4) has been related to the definition of a middleware in order to allow the three modules to exchange information, to acquire information from sensors, and to control devices in the environment. To this aim, an analysis of the state-of-the-art frameworks has been carried out, starting from the required characteristics:

- *Resources scalability* (to comply with devices with limited computational and power resources);
- *Interoperability* (to support heterogeneous devices);
- *Devices scalability* (to retain good performance even with a large number of devices connected);
- *Mobility* (to allow for devices to change their location, thus possibly leading to a reconfiguration of the network, while being connected to the others);
- *Heterogeneity abstraction* (to hide lower-level complexity from higher-level);
- *Modularity* (to make the middleware itself more flexible and maintainable);
- *Adaptability* (to deal with random and unpredictable changes, e.g., portions of the network suddenly becoming unavailable);
- *Security and privacy* (to protect the data shared over the network from being accessed by unauthorised applications);
- *Tools and algorithms* (to solve typical robotic problems wasting resources on problems that have been already faced by the community);
- *Real-time and efficiency* (to guarantee compliance with real-time constraints related to software, hardware and communication mechanisms).

In the past decades, many middleware for robot control with some of the required characteristics have been developed. For the sake of brevity, the reader is referred to complete surveys of robotics software frameworks such as [9], [10]. Here it will be only remarked that, among all possible frameworks, universAAL [11] (uAAL) has been chosen for the CARESSES architecture, being one of the most widespread open software platforms for smart environments. In particular, it has been preferred to more popular frameworks such as ROS [12] in this context since it has been specifically designed for Ambient Assisted Living applications in smart homes[6]. Given these premises, it is a good candidate to become a standard for the development of assistive solutions in smart ICT environments that include distributed and wearable sensors, devices, robots, and multimodal user interfaces. Moreover, it meets almost all the requirements listed before (except *Tools and algorithms* and *Real-time*, that however are not mandatory for the CARESSES purposes) and it implements three different communication policies: Publish/subscribe, Service-oriented architecture and Semantic ontology approaches.

Being an ontology-based middleware, uAAL allows designers to describe the semantics of the exchanged messages in a unique, machine understandable, way.

C. Messages for Inter-Module Communication

As already mentioned, the three CARESSES components exchange messages with each other: categories of exchanged messages are depicted in Figure 2, and briefly listed below.

- *D1. Planning info*. Required information to generate a plan depending on the cultural context (CKB to CSPEM).
- *D2. Actions and Topics*. Required information to execute actions and switch to the desired conversation topics depending on the plan and the cultural context (CKB to CSPEM).
- *D3. Cultural Norms and Preferences*. Constraints or preferences on the admissible states (CKB to CSPEM).
- *D4. Goals*. Persistent goals that are stored in the Cultural Knowledge Base (CKB to CSPEM).
- *D5. User input*. Inputs acquired through speech recognition and through the robot's tablet (CAHRIM to CKB and CSPEM).
- *D6. Current State*. Current state of the user, the environment and the robot (CAHRIM to CSPEM).
- *D7. Action and topic request*. Actions to be executed (CSPEM to CAHRIM).
- *D8. User habit - personality*. Detected user habits, preferences or personality traits (CAHRIM to CKB).

To enable modules developed by different researchers to exchange message with a shared semantics (i.e., to avoid ambiguity in the format and interpretation of messages), a handbook describing the type, subtypes, and format of all messages in details has been prepared as a support to the functional architecture (months M3-M4) and made available to developers. Also, based on the universAAL framework and its ontology-based communication mechanisms, a "software skeleton" has been prepared (Figure 2 below). The software skeleton includes the uAAL communication buses as well as three mock-up modules CAHRIM, CKB, CSPEM: these modules are initially empty, and can be used by researchers as starting points to iteratively develop, test, and refine their

[6]universAAL has resulted from a consolidation process conducted within the EU-FP7 project, `http://universaal.sintef9013.com/index.php/en/`

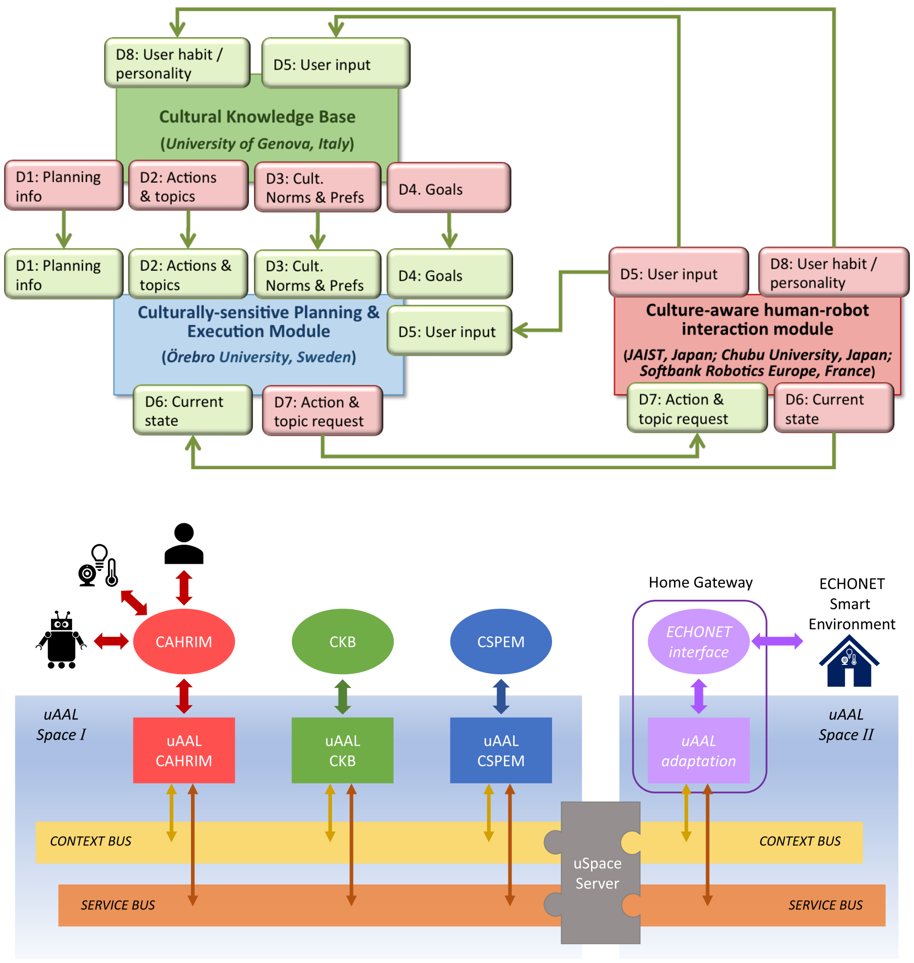

Fig. 2. Above: CARESSES architecture. Below: implementation in uAAL. An additional fourth component, *Echonet interface* represents the framework used by JAIST, one of the Japanese partners, for the management of the smart environment, integrated within the uAAL framework.

solutions, by guaranteeing that all the developed solutions are integrated since the first months of the project. Even more, the software skeleton constrains CAHRIM, CKB, CSPEM to exchange only messages whose type is coherent with the functional architecture.

D. Collaboration and integration tools

The process of software integration during the first year of CARESSES has taken inspiration from the *DevOps* concept, i.e., the boost to fuse together software development (Dev) and software operation (Ops), whose techniques have been mainly applied to the IT commercial domain. In particular, some of the main methodologies descending from the *DevOps* concept are:

1) Continuous Integration: Continuous Integration (CI) is a software development practice consisting mainly in frequent, possible daily, integration of the work of the single developer (ideally this practice leads to multiple integrations per day) [2], [13]. This methodology is based on some key practices [14], among which the most relevant are the usage of a single source repository, tools for automated builds and tests, daily commits to the repository.

The idea underlying CI is that the process of integrating software on a daily basis, without using different branches and by exploiting tools for automated builds and tests can greatly help in reducing risks of deferred integration, in detecting bugs and in having frequent deployments, increasing therefore the link between customers and developers.

Obviously it requires a quite experienced team and some time for getting acquainted with the tools and procedure.

2) Continuous Delivery: Continuous Delivery (CDE) extends the concept of CI, focussing on the practice of frequently delivering quality software [15]. To this aim, the application should be always in a production-ready state, after successfully passing automated tests and quality checks. Among its key concepts, the most important are the implementation of a good branching strategy for software repositories and the usage of distributed testing infrastructures.

Some of the key aspects of the methodologies described before have then been adopted for software integration within the project, taking into account the peculiar characteristics of a joint scientific research project with respect to an IT commercial project (e.g. different backgrounds and experience of the developers involved, absence of real customers). Among

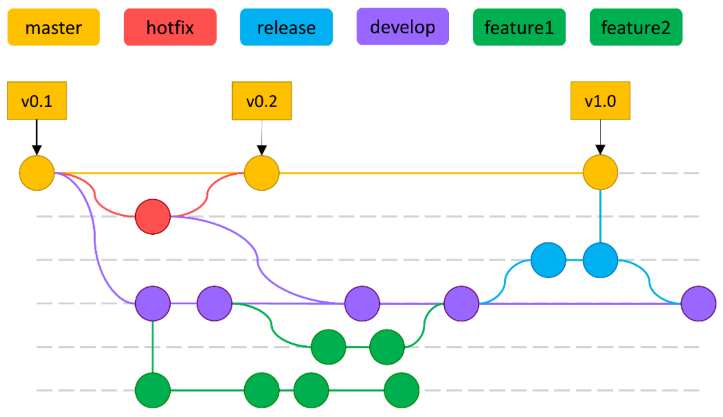

Fig. 3. Gitflow workflow.

the key concepts adopted, the integration of software starting from the early stage, the usage of a single source repository, periodic commits, delivery of releases (this corresponds to the implementation of *integrated demos*) and the adoption of a multi-branching strategy follow the *DevOps* general guidelines.

Going more into details, the developed software has been shared among CARESSES partners since the beginning of the project through a Git repository, set up on GitLab. The software repository's organization has been structured to reflect the CARESSES software's architecture, with a hierarchical organization considering the three modules CKB, CSPEM, CAHRIM and a folder for universAAL components. Additionally, the repository contains a folder to host the documentation and the auxiliary files which are needed to run the official, past and upcoming, demos. The idea is that everything, including third party libraries and install scripts, should be added to the repository, so that anyone should be able to fully build the system, even on a virgin machine, by simply doing a checkout from the repository.

In order to successfully work on the same shared software hosted on the repository, a standard unique workflow, based on a multi-branching strategy, to be followed by all partners has been adopted: the Gitflow workflow [16].

In the Gitflow workflow (Fig. 3), two main branches are present: *master* and *develop*. The *master* branch stores the official release history, and the *develop* branch serves as the testbed for new features and functionalities. To work on new features, each developer must fork a *feature* branch from *develop*. Each new feature should reside in its own branch, which can be pushed to the shared repository for backup/collaboration. When a feature is complete, the developer merges the *feature* branch into *develop* and tests whether the newly added feature works properly or not. Once *develop* has acquired enough features for a release (*integrated demo*) and each developer has tested and documented the new features they have developed, a *release* branch is forked from *develop*. On the *release* branch, all the partners test the whole software but no new features can be added (only bug and documentation fixes). Once the software is ready and working, the *release* branch is merged into *master* and the commit is tagged with a version number. In addition, *release* is merged into *develop*. A multibranch-based workflow such as the one used enables to split out the development as per specific features, keeping the master branch safe while working on single code aspects.

A Wiki has been created in the software repository to host the necessary documentation (guidelines and tutorials), to sum up the goals expected in each demo and to officially report the testing-state of the ongoing *integrated demo*.

However, adopting common guidelines to work on the same repository is not sufficient to successfully collaborate on the development of the same software. Therefore, importance has been given to the use of specific communication channels in order to help developers in easily discuss and take low-level decisions. To this aim, Slack[7], a set of team collaboration tools meant for organizational communication, has been adopted and used, for example, to share pieces of code and architecture schemes, give feedbacks, notify about software components updates and give more detailed explanation on specific developed features by exploiting the topic-oriented multichannel strategy.

Finally, in order to foster the integration activities, two *integrated demos* have been planned during the first year (in July 2017 – M7, and November/December 2017 – M11/12), with an increasing number of features.

E. Design of the integrated demos

As mentioned before, two *integrated demos* (software releases) have been planned and implemented during the first year of the project.

1) Demo 1 (June 2017): The first demo has been mainly focused on testing the data exchange through the uAAL framework, by purposely ignoring the specific objectives of CARESSES (achieving a culturally competent robotic behaviour). Thus, with the main purpose of testing the process for collaborative software development and integration, the following three tasks have been planned:

[7]`https://slack.com/`

- Approach the user and greet him/her (the task does not imply interaction with the user or with the environment);
- Approach the user and propose him/her to place a video call (the task implies only interaction with the user);
- Approach the user and let him/her ask for the room temperature, retrieved with sensors in the environment (the task implies interaction with the user and with sensors in the environment).

The demo has been carried out with a limited set of actions, that are available to CSPEM for planning and implemented in CAHRIM for execution: *ApproachUser*, *GreetUser*, *PlaceVideoCall*, *ListenUserRequests* and *SayTemperature*. All tasks require a strict integration between the three CARESSES components. Let us analyze for instance the first task. At the beginning, CSPEM (developed by Örebro University) requires messages of type D1 and D2 (i.e. available planning operators and actions) from CKB (developed by University of Genova); then, after a while, CKB sends a message of type D4 (current goal: *Greet the user - now*). Once received the goal, CSPEM computes a plan made of two actions: *ApproachUser* and *GreetUser* and sends a message of type D7 on the uAAL bus. The message is received by CAHRIM that executes the action *ApproachUser* (developed by JAIST), while periodically sending a message D6 (the action execution state) through the uAAL bus. As soon as the message D6 notifies that the action has been executed, the planner sends a second message D7 with the second action to be executed (*GreetUser*). Again CAHRIM executes the action, sending periodic information to CSPEM, and notifying the end of the action. Finally CSPEM declares that the goal *Greet the user - now* has been reached and waits for another goal or user requests. For all tasks, UML Sequence Diagrams have been prepared and made available to all partners in the common repository.

The other two tasks have been implemented in a very similar way. However, the third task requires also communication between CAHRIM and a module for the management of temperature sensors (which - in our case - require bridging universAAL with the iHouse ECHONET environment, a duplex apartment at JAIST fully embedded with sensors and actuators (Fig. 2) [17]).

2) Demo 2 (December 2017): The second demo has been mainly focused on (i) adding cultural aspects, and on (ii) increasing the number of actions available for planning and execution

In relation to point (i), the CKB has been populated by encoding culturally competent behaviours related to three reference cultures considered in the project: English, Indian and Japanese (thus, the demo can be executed by simulating persons belonging to these three different nationalities). More specifically, the core of the CKB encodes elements that may play a key role in socially assistive robotic scenarios: in relation to the three different culture, different goals to be suggested, different actions with different parameters (e.g., distance from the person during interaction, navigation speed, volume), different norms, and finally conversation topics with different sentences and *likeliness* values (i.e., representing the probability that a person belonging to a specific culture may have a positive attitude towards that topic of conversation).

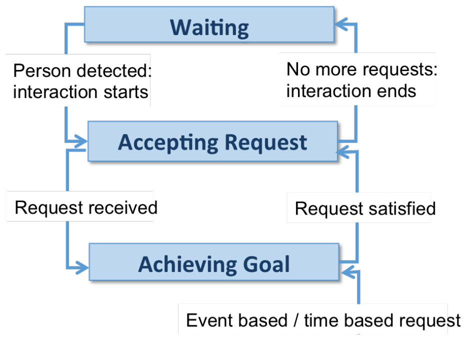
Waiting
Person detected: interaction starts
No more requests: interaction ends
Accepting Request
Request received
Request satisfied
Achieving Goal
Event based / time based request

Fig. 4. General scheme of the second *Integrated demo*.

Please notice that all this information have been encoded at a general cultural level: this means that, in absence of person-specific knowledge, the robot will presume that information generally valid for a culture holds for the specific person. However, to avoid stereotyped representations, it will always ask the person for confirmation: as soon as the robot acquires specific knowledge (through observation or verbal interaction), it will create new instances in the ontology that will be used as new references for the interaction[8].

Regarding (ii), the demo has been modelled as a state machine with three phases (Fig. 4):

- *Waiting*. The robot waits for somebody to enter the room. When the robot detects a person and/or a person asks the robot that he/she wants to start the interaction, the robot switches to the state *Accepting request*.
- *Accepting Request*. The robot is waiting for requests, that can either be given through verbal interaction or through the robot's tablet. If the user makes a request, the robot switches to the state *Achieving Goal*. Otherwise, if the user does not make any request for a long time, or says explicitly that he/she wants to terminate the interaction, the robot switches to the state *Waiting*.
- *Achieving Goal*. The robot performs the sensorimotor and verbal actions required to achieve the goal. After achieving the goal, the robot goes back to the state *Accepting Request*.

Moreover, in order to enrich the interaction with the user, new actions have been added to the system (once again, planning info are initially sent by CKB to CSPEM as messages of type D1 and D2; actions are executed by sending proper messages of type D7 to CAHRIM; a feedback to CSPEM about the current system state is sent back as a message of type D6):

- *Chitchat* (developed by University of Genova). Please remind that the CKB stores possible conversation topics, together with keywords that trigger the dialogue, sentences and questions that the robot may say and finally likeliness values for each considered culture. All these

[8]Elements that are culturally relevant have been defined in accordance to the guidelines provided by Transcultural Nursing experts, note 5.

elements and the hierarchical organization of the ontology allows the robot to follow some basic strategies for a more involving dialogue pattern. For instance, if an English man says something about sport, the robot will probably ask him if he likes sport and then (in case of a positive answer) if he likes watching rugby or soccer on TV (since these sports are very popular in the UK). With the same rationale, when talking about food with a Japanese person, the robot will initially assume that the person prefers Japanese food and, upon confirmation, it may ask something about ramen or sushi. Please notice that this does not prevent an Indian person from speaking about Japanese food or rugby, given that the related triggering keywords are pronounced. During the action, the robot acquires person-specific information that update the CKB (messages D5).

- *AcceptRequest* (developed by SoftBank Robotics and University of Genova). The robot acquires a request from the user, using verbal interaction or through the tablet. The most probable requests for each culture (or for each individual, if this information is in the CKB) are passed as cultural parameters, so that the robot can suggest tasks it is ready to perform for the person in a culturally competent way. When the person has selected a goal, a message D5 encoding the user's request is produced and sent to CSPEM.
- *Greeting* (developed by SoftBank Robotics). Three actions have been implemented: *GreetByWaving* (the robot waves the hand), *GreetNamaste* (the robot makes the Namaste gesture) and *GreetBow* (the robot bows). When the robot has to greet the person, one of them is selected by taking cultural information into account.
- *SetReminder* (developed by University of Genova) The action includes a dialogue pattern to set the task and the time of the reminder. Then it generates a proper message D5 that is received by CSPEM to produce a proper goal to be performed in the future (notice that this information may also be stored in the CKB through a message D8 as it corresponds to a habit). As for the *AcceptRequest* action, the most probable goals to be reminded for each culture are passed as parameters of the action and used by the robot as suggestions for the user.
- *ApproachObject* (developed by JAIST). The robot approaches the requested object by avoiding obstacles.
- *PlayMusic* (developed by JAIST). After a short dialogue pattern for selecting the music genre, the robot plays music on its tablet.
- *ReadTemperature* (developed by JAIST). The robot retrieves the temperature from sensors in the environment by communicating with the iHouse ECHONET network, and displays / says its value.
- *DisplayWeatherReport* (developed by JAIST). The robot displays on the tablet weather information streamed from the Internet and reads them.

Please notice that adding a new action requires a high level of integration of the system: indeed, new actions should be modelled in the CKB (and linked to their cultural and formal parameters), in CSPEM (described in the planner formalism and associated to action operators) and in CAHRIM (with the actual description of the sensorimotor behaviour necessary for implementing the action). Thus, the integration approach based on daily commits and branching strategy turned out to be fundamental for the success of this *integrated demo*.

F. Practical Implementation

The two demos have been tested in four different locations (University of Genova, Italy; Örebro University, Sweden; JAIST, Japan; SoftBank Robotics, France), thus allowing for obtaining a useful feedback on the software integration. Since the developed system may allow the user to freely interact with the robot (within the limits of the available and feasible tasks), specific steps have been defined to guarantee an equal execution of the demos in every location. As a result, all partners have been able to execute the whole set of actions, with their specific cultural parameters, through verbal interaction with the robot, achieving an identical behaviour in their home laboratories within the end of the first project's year.

Video footages of the experimental sessions have been recorded and used as a very practical way to compare the obtained results and bring out unforeseen differences (Fig. 5). An edited video, showing all partners interacting with the robot as fictional English, Indian and Japanese characters is attached to this article[9].

IV. CONCLUSIONS

This article discusses the importance of a solid integration strategy in the context of joint international research projects, on the heels of common practices followed in the commercial IT domain. As a case study, the CARESSES project and the integration work performed since the beginning of the project by the authors have been deeply described and analyzed.

Generally speaking, the successful design and implementation of two *integrated demos* shows that adopting a fully collaborative approach since the early stages of the project has clear advantages, even in a context different from the IT commercial domain: (i) periodic releases of software that can be therefore tested by all partners; (ii) immediate detection of bugs and deviations from workplan; (iii) continuous refining of the single software components.

Moreover, while automated tools for parallel testing have not been used, the possibility of experimenting the whole system in four different locations (University of Genova, Italy; Örebro University, Sweden; JAIST, Japan; SoftBank Robotics, France), with different hardware, helped significantly to define a minimum set of third-party software components (that have been added to the common repository) necessary for running both demos. Given the peculiar context of a social robotic project, video footages of the experimental sessions have been used not only as a tool for disseminating

[9] `https://www.youtube.com/watch?v=QNkpjzwLri0`

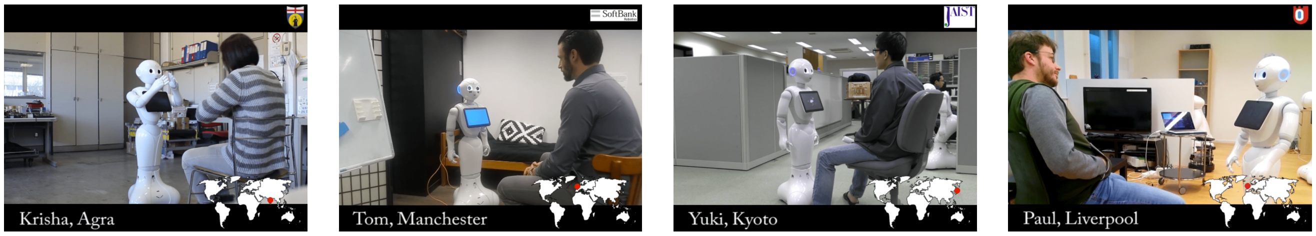

Fig. 5. Snapshots of the demonstrations, performed by all technical partners of the CARESSES project. Names and nationalities are simulated for the demo purposes.

the results, but also as an internal method for evaluating the overall performance across different locations.

On the other side, the implementation of such an approach requires a not negligible amount of time for defining clear guidelines, for getting all developers acquainted with the methodology and for implementing some simple procedures to test the system: it can be pointed out that this may slow down the development of single software components. However, this approach guarantees the compatibility of each component with the others, since the interface with the rest of the system is fully defined. This means that developers can rely on a solid structure for future work, integrating features without the risk of modifying big portions of the code for integration purposes at the very end of the project.

ACKNOWLEDGMENT

This work has been supported by the European Commission Horizon2020 Research and Innovation Programme under grant agreement No. 737858, and from the Ministry of Internal Affairs and Communication of Japan. The authors acknowledge the contribution of other CARESSES partners: University of Bedfordshire (UK), Middlesex University (UK), Advinia HealthCare, (UK), Nagoya University (Japan).

REFERENCES

[1] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909-3943.

[2] Fowler, M., & Foemmel, M. (2006). Continuous integration. Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf, 122, 14.

[3] David, V., Erik, B., Paul, H., Serge, T., & Tom, Z. (2015). An Architecture-oriented Approach to System Integration in Collaborative Robotics Research Projects. An Experience Report. Journal of Software Engineering in Robotics, 6(1), 15-32.

[4] Bubeck, A., Weisshardt, F., Sing, T., Reiser, U., Hgele, M., & Verl, A. (2012, December). Implementing best practices for systems integration and distributed software development in service robotics-the Care-O-bot® robot family. In System Integration (SII), 2012 IEEE/SICE International Symposium on (pp. 609-614). IEEE.

[5] Bruno, B., Chong, N. Y., Kamide, H., Kanoria, S., Lee, J., Lim, Y., ... & Saffiotti, A. (2017). Paving the Way for Culturally Competent Robots: a Position Paper. In 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN).

[6] I. Papadopoulos (2006). Transcultural health and social care: development of culturally competent practitioners. Elsevier Health Sciences.

[7] Wang, L., Rau, P. L. P., Evers, V., Robinson, B. K., & Hinds, P. (2010, March). When in Rome: the role of culture & context in adherence to robot recommendations. In Proceedings of the 5th ACM/IEEE international conference on Human-robot interaction (pp. 359-366). IEEE Press.

[8] Guarino, N. (Ed.). (1998). Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy (Vol. 46). IOS press.

[9] Iñigo-Blasco P., Diaz-del-Rio, F., Romero-Ternero, M. C., Cagigas-Muiz, D., & Vicente-Diaz, S. (2012). Robotics software frameworks for multi-agent robotic systems development. Robotics and Autonomous Systems, 60(6), 803-821.

[10] Recchiuto, C. T., & Sgorbissa, A. Post-disaster assessment with unmanned aerial vehicles: A survey on practical implementations and research approaches. Journal of Field Robotics. DOI: 10.1002/rob.21756

[11] Hanke, S., Mayer, C., Hoeftberger, O., Boos, H., Wichert, R., Tazari, M. R., ... & Furfari, F. (2011). universAAL-an open and consolidated AAL platform. In Ambient assisted living (pp. 127-140). Springer, Berlin, Heidelberg.

[12] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[13] Ståhl, D., & Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. Journal of Systems and Software, 87, 48-59.

[14] Meyer, M. (2014). Continuous integration and its tools. IEEE software, 31(3), 14-16.

[15] Pathania, N. (2017). Elements of Continuous Delivery. In Pro Continuous Delivery (pp. 1-21). Apress, Berkeley, CA.

[16] Driessen V., A successful Git Branching model `http://nvie.com/posts/a-successful-git-branching-model/`, Last accessed on 2018-03-20.

[17] Lim, Y., Lim, S. Y., Nguyen, M. D., Li, C., & Tan, Y. (2017, June). Bridging between universAAL and ECHONET for smart home environment. In Ubiquitous Robots and Ambient Intelligence (URAI), 2017 14th International Conference on (pp. 56-61). IEEE.