ICRA-07 Workshop on Network Robot Systems
IEEE Int Conf on Robotics and Automation. Rome, Italy, April 2007

# Reactive Self-Configuration of an Ecology of Robots

Marco Gritti, Mathias Broxvall, Alessandro Saffiotti
AASS Mobile Robotics Lab
Örebro University, Örebro, Sweden
{mgi,mbl,asaffio}@aass.oru.se

***Abstract*— The field of ubiquitous robotics is burgeoning, and different brands of massively distributed heterogeneous robotic systems are being proposed and applied to several domains. The strong added value of these systems comes from their potential ability to dynamically self-configure, by changing the form of their cooperation to adapt to a given task or situation. In face of this, no satisfactory solution exists to the problem of how such a system should self-configure. In this paper, we explore a reactive approach to self-configuration inspired by ideas from the field of semantic web services. We illustrate our approach on a specific type of ubiquitous robot system, called a PEIS Ecology. We show experiments in which our approach autonomously generates a configuration to perform a cooperative navigation task, and dynamically changes this configuration when one of the components fails.**

## I. Introduction

The field of robotics is witnessing a rapid change, which many observers compare to the one occurred twenty years ago in the field of computing. Robotic technologies begin to be employed in relatively inexpensive devices and consumer products, and they slowly start to appear in our homes, offices, and workshops. Soon many of the devices around us will incorporate robotic technologies, although these will probably not look like today's robots more than a mobile phone looks like yesterday's mainframe computers. Most of these devices will be networked and they will have the potential to share their information and coordinate their actions. This potential is believed to provide the basis to boost the capability and applicability of robotic technology far beyond its current limits.

The above vision is becoming rather popular in our field, and it has been spelled out under different names, including network robot systems [1], intelligent spaces [2][3], sensor-actuator networks [4], ubiquitous robotics [5], and PEIS-Ecology [6]. In this paper, we shall generally refer to this vision as "ecology of robots". One of the tenants of this vision is that a major breakthrough in service and home robotics can be obtained by replacing the idea of building one extremely competent isolated robot acting in a passive environment by a network of cooperating robotic devices embedded in the environment.

According to this vision, the strong added value of an ecology of robots comes from the ability of these robots to integrate their functionalities in a collaborative way, and to automatically organize, or *self-configure*, this collaboration. Self-configuration is needed since an ecology of robots is intrinsically dynamic (robots may join and leave the ecology at any time), and is exacerbated by the high heterogeneity of such system. Although a lot of work has been done recently on the principles and techniques for automatic self-configuration in autonomous robotics, as well as in other areas of computer science (e.g., ambient intelligence [7], web service composition [8], autonomic computing [9]), no satisfactory solutions exist yet.

In this paper, we explore a reactive approach to self-configuration of an ecology of robots. In this approach we borrow some ideas from the field of Semantic Web Services, and adapt them to the different needs of our domain. The main ingredients of our approach are:

- *Formal descriptions* of functionalities, so they can be exported to the ecology and automatically processed;
- A framework for *discovery and composition* that allows robots to find exported functionalities compatible with their needs, and allows the ecology to configure at run-time for a given task assembling matched functionalities from different robots;
- A mechanism of *semantic interoperability* to match functionalities from heterogeneous devices according to a unified ontological classification.

In order to illustrate our approach, we apply it to our specific brand of an ecology of robots, called a PEIS-Ecology. The approach, however, should be applicable to other approaches to ubiquitous or ecological robotics as well.

The rest of this paper is organized as follows. In the next section we recall the basic notions of a PEIS-Ecology. In Section 3 we describe the main ingredients needed for self-configuration listed above. In Section 4 we provide a concrete instance of a reactive configuration algorithm. In Section 5 we show an example of self-configuration run on our experimental testbed.

## II. The PEIS-Ecology Approach

The concept of PEIS-Ecology [6] puts together insights from the fields of ambient intelligence and autonomous robotics to generate a radically new approach toward the inclusion of robotic technologies in everyday environments. In this approach, advanced robotic functionalities are not achieved through the development of extremely advanced robots, but through the cooperation of many simple robotic components.

The concept of PEIS-Ecology builds upon the following ingredients (see [10], [11] for more information).

First, any robot in the environment is abstracted by the *uniform notion* of PEIS (Physically Embedded Intelligent System): any device incorporating some computational and

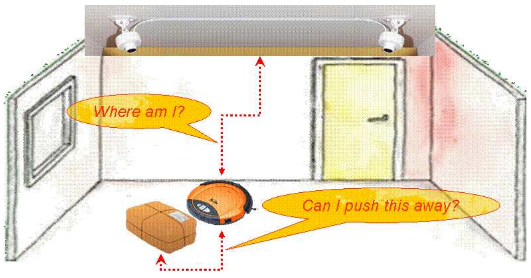


Fig. 1. A simple example of cooperation in a PEIS-Ecology.

communication resource and possibly able to interact with the environment through sensors and/or actuators. A PEIS can be as simple as a toaster and as complex as a humanoid robot. In general, we define a PEIS to be a set of inter-connected software components residing in one physical entity. Each PEIS-component may include links to sensors and actuators, as well as input and output ports that connect it to other PEIS-component in the same PEIS or in other PEIS.

Second, all PEIS are connected by a *uniform communication model*, which allows the exchange of information among the individual PEIS-components, and can cope with their dynamic joining and leaving the ecology. This model has been implemented in a specific middleware, the PEIS-kernel, which implements a *distributed tuple space* over a P2P network. This tuple space is used as the primary communication mechanism between PEIS-components by creating and consuming tuples.

Third, all PEIS can cooperate by a *uniform cooperation model*, based on the notion of linking PEIS-components: each participating PEIS can use functionalities from PEIS-components of other PEIS in the ecology in order to compensate or to complement its own. In our middleware this cooperation comes from *subscriptions* to tuples, where a consuming PEIS-component automatically receives the tuples created by a producing PEIS-component run within any other PEIS.

As an illustration of these concepts, consider an autonomous vacuum cleaner. By itself, this simple PEIS does not have enough sensing and reasoning resources to assess its own position in the home. But suppose that the home is equipped with an overhead tracking system, itself another PEIS. Then, we can combine these two PEIS into a simple PEIS-Ecology, in which the tracking system provides a global localization functionality to the cleaning robot, which can thus realize smarter cleaning strategies. (See Figure 1.) Suppose further that the cleaner encounters an unexpected parcel on the floor. It could push it away and clean under it, but it needs to know its weight to decide if the parcel can be pushed. If the parcel is equipped with an IC-tag, it can act as a PEIS and communicate this information directly to the cleaner.

We define a PEIS-*Ecology* to be a collection of inter-connected PEIS, all embedded in the same physical environment. We call *configuration* of a PEIS-Ecology the set of connections between PEIS-components within and across the

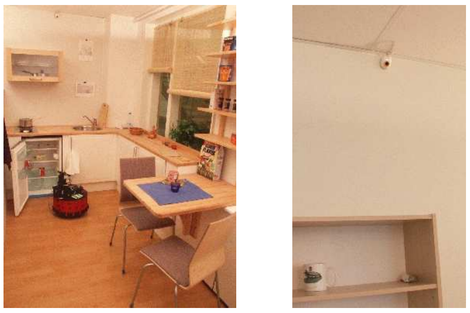


Fig. 2. The PEIS-Ecology testbed. Left: the kitchen. Right: detail of one of the ceiling cameras.

PEIS in the ecology. More precisely, a configuration in which a PEIS-Ecology operates for an interval in time is determined by:

1) the collection of all the *running* PEIS-components;
2) the collection of all the *parameters* that are set for each PEIS-component;
3) the collection of all input-output *connections* between PEIS-components.

Note that the same ecology can be configured in many different ways depending on the current context. Relevant contextual aspects here include the current goals, situation, and resources. In our middleware a configuration corresponds to a set of *subscriptions* between components.

In order to validate the concept of PEIS-Ecology, we have constructed the PEIS-home: an apartment-like environment that incorporates a number of PEIS. The PEIS-home –see Figure 2– is used as physical demonstrator in which to run all the experiments related to the PEIS-Ecology, included the one described in section V.

## III. SELF-CONFIGURATION

The need for a mechanism of self-configuration is motivated by the intrinsic nature of the PEIS-Ecology. This is a distributed system of heterogeneous software components whose collaboration patterns are not fixed at design time. A PEIS-Ecology is indeed a dynamic software environment, in which components can join or leave at run-time. As soon as a PEIS physically enters the ecology, the components it carries can possibly establish connections with the components in the rest of the ecology. When a PEIS leaves the ecology, the connections should be seamlessly unleashed. Moreover, even when the PEIS and PEIS-components are fixed in the ecology, their collaboration patterns could change according to the different tasks the ecology can perform, or the different operating conditions: the vacuum cleaner could use the overhead tracking system when located inside the field of view of the ceiling cameras, and could use other resources for localization when outside their field of view. All this implies that the PEIS-Ecology should possess the ability to automatically determine its configuration, and to modify it when needed.

Heterogeneity of the software applications, i.e. the PEIS-components, that run on this platform is another issue which is tightly related to the problem of self-configuration.

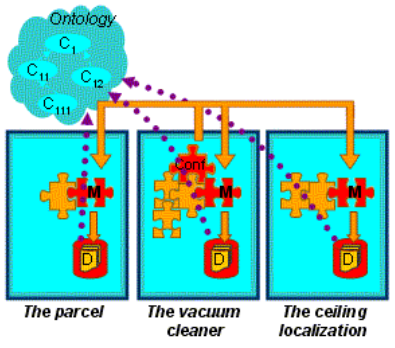


Fig. 3. The self-configuration framework of a PEIS-Ecology.

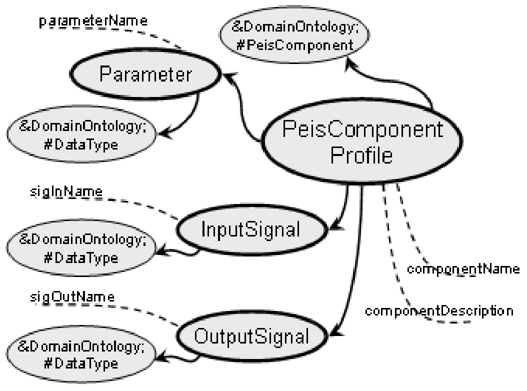


Fig. 4. The PEIS Profile ontology.

The PEIS-kernel middleware provides the developer with a uniform programmatic interface to enable data transfer, and moves the interoperability problem to the semantic level. Establishing connections between components becomes a problem of determining which component provides functionalities or information which are semantically compatible with the requested ones. We claim that this computation can be done automatically. The need for semantic interoperability motivates the choice of adopting tools and techniques from ontology engineering, as described in section III-C.

Figure 3 illustrates our framework to solve the self-configuration problem: every PEIS is provided with a local directory of formal descriptions `D` of available components and with a special component `M` that can access the descriptions and propagate them to the rest of the ecology. This component is also responsible for starting and stopping other local components on demand. Any PEIS can be equipped with a special component, here denoted as `Conf`, that is capable of retrieving the descriptions and computing a meaningful configuration upon the information stored in them. Various components of type `Conf` can exist in the ecology, solving the configuration problem with possibly different techniques; section IV will illustrate a concrete implementation of a `Conf` PEIS-component. Figure 3 also shows that both the component descriptions and the `Conf` PEIS-component have access to the same common ontology. In the following subsections we are going to detail the important aspects of this framework: formal description (`D`), discovery and configuration (`M` and `Conf`), and semantic interoperability.

## A. *Formal description of* PEIS*-components*

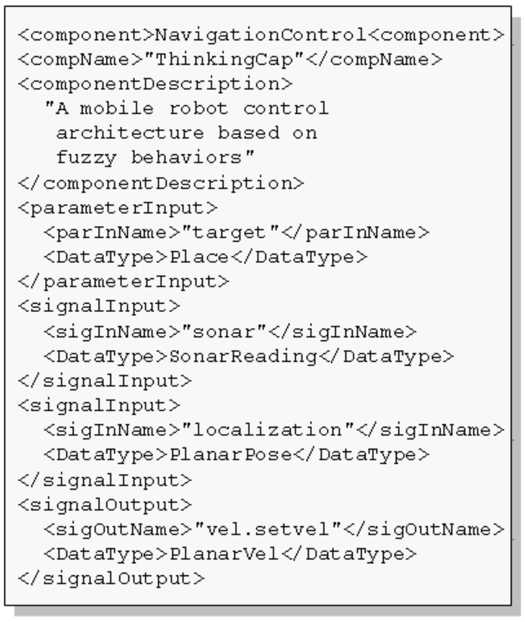
A graphical representation of a PEIS-component formal description, modeled as an ontology, is given in Figure 4. A PEIS-component description is an *instance* of the `Profile` class. The serialization of a profile of a PEIS-component in a specific ontology language is the formal component description used for automatic configuration. This serialization can also be referred with the term "advertisement", since a PEIS-component is *published* into the PEIS-Ecology by means of sending this serialization to the other PEIS-components.

As an example, consider the XML-like serialization in Figure 5, which represents the advertisement of a mobile robot navigation control PEIS-component. The profile specifies that the category of the advertised component is `NavigationControl`. This is useful to allow applications that are looking for other kinds of components, e.g. manipulator control systems, to discard this advertisement, just ignoring the rest of it. The advertisement specifies that in order to run correctly this PEIS-component should be continuously fed with two input signals: the `sonar`, and the `localization`. The name of the input signals are just tags. The semantically interesting quantity is the type of the signals. They are respectively `SonarReading` and `PlanarPose`, and they refer to classes defined in the domain ontology. The same can be said about the unique output signal of the component. The unique parameter of the component is the destination, of type `Place`, and it represents the final destination to which this component is supposed to move the robot.

This example helps to point out the main idea that characterizes PEIS-components profiles with respect to Web Service profiles, as they are modeled in OWL Services (OWL-S) [12]. Generic services are usually conceived as method invocations with input and output parameters. Instead, when describing PEIS-components it is fundamental to distinguish between parameters and signals: the former are a quantities assigned just once per execution; the latter are quantities that need to be continuously updated during the execution, and special constraints like sampling rate and precision can be associated to them. This difference is clear for `target` as opposed to `localization`: the final destination is an input parameter of the function-like call to `NavigationControl`, while the localization information has to be provided continuously during the whole execution of the function, with strict time constraints.

## B. *Discovery and composition of* PEIS*-components*

As typical peer-to-peer platform, the PEIS-Ecology lists decentralization among its main requirements. This excludes the possibility of having a centralized directory of component advertisements. The common solution, also adopted in other decentralized architectures [13][7], is to deploy the advertisements of each component on the same site the component is run. Component advertisements are propagated upon request to any application that needs to perform some



```
<component>NavigationControl<component>
<compName>"ThinkingCap"</compName>
<componentDescription>
  "A mobile robot control
   architecture based on
   fuzzy behaviors"
</componentDescription>
<parameterInput>
  <parInName>"target"</parInName>
  <DataType>Place</DataType>
</parameterInput>
<signalInput>
  <sigInName>"sonar"</sigInName>
  <DataType>SonarReading</DataType>
</signalInput>
<signalInput>
  <sigInName>"localization"</sigInName>
  <DataType>PlanarPose</DataType>
</signalInput>
<signalOutput>
  <sigOutName>"vel.setvel"</sigOutName>
  <DataType>PlanarVel</DataType>
</signalOutput>
```

Fig. 5. The advertisement of a navigation control system.

kind of computation on them, for example, to verify if the described component fits into an intended configuration.

A PEIS-*Meta* component is installed on each PEIS. This component has access to a local directory which stores the advertisements of the components that can be run on that PEIS. This PEIS-Meta-component is also responsible for starting and stopping local components on demand, e.g. when a configuration is deployed. Any other PEIS-component of the same PEIS-Ecology can *query* the PEIS-Meta-component about the presence of PEIS-components providing the needed functionalities. Queries can be matched with advertisements in a purely syntactical way, or more complex reasoning techniques can be used, provided that the advertisements store semantic descriptions of components.

As sketched in Figure 6, the application that issues the queries to the PEIS-Meta-component is the same that performs the automatic ecology configuration. The reactive configuration application that will be described in the next section belongs to this category. As shown in the figure, after the configuration application has broadcasted the queries about the needed functionalities (step 1), and after the various PEIS-Meta-components have answered with relevant component advertisements (step 2), the configuration application evaluates the feasibility of a configuration given the advertisements it has received (step 3). The first two steps can be altogether referred as "discovery" of PEIS-components, while step three is referred with the term of "composition" of PEIS-components. This concepts, together with the concept of advertisement are inherited from the field of Web Services, which inspired this work. Figure 6 shows that finally, if an admissible configuration was found, the configuration application performs the corresponding connections between input and output signals of the selected PEIS-components (step 4). The connections are made by invoking the underlying middleware.

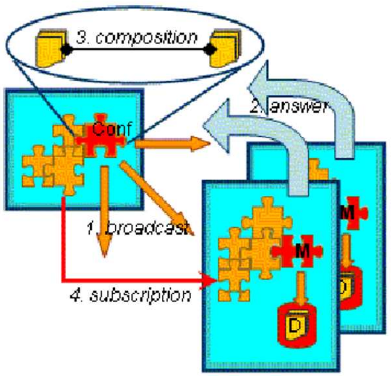


Fig. 6. Discovery and composition of PEIS-components.

The problem of *composing* PEIS-components into a meaningful configuration can be seen as a problem of *search* in the space of all possible configurations for a single *admissible* configuration. This search can be conducted through plan-based techniques in the full space of possible configurations or into a subset of it, built upon some heuristic function.

In section IV we propose the use of a reactive configuration algorithm: this works just on the set of received advertisements. Moreover, not all of the possible configurations built upon the set of components whose advertisements have been received are explored by this algorithm, but the search is *constrained* to the exploration of just a subset of possible configurations using a *template* configuration. This is a specification of the categories of components that constitute a correct configuration, and of all meaningful connection patterns within these categories. The template configuration can be built in the logics of the specific application component.

## C. *Semantic interoperability of* PEIS-*components*

Evaluating the correctness of a given configuration is a problem of semantic matching between component descriptions. The burden of semantic matching is shared in our framework between the PEIS-Meta-component and the configuration component. In extreme cases, semantic matching could be done just at the side of the configuration application, or just at the side of the PEIS-Meta-components. In order to answer complex queries, the PEIS-Meta-component has to be provided with basic reasoning abilities. More precisely, the PEIS-Meta-component should be provided with an algorithm able to determine if between the components listed in its knowledge base there exists some that instantiates the requested category, provides a functionality that instantiates the requested functionality, and is compliant to the requested properties.

In the current implementation the PEIS-Meta-component installed on every PEIS does not perform any reasoning on queries and advertisements, but simply answers to the queries by propagating all the advertisements stored in the local directory. However, in future work we expect to implement more advanced schemes for this, in which the PEIS-Meta-component features some basic reasoning ability, obtained by tailoring some well known semantic matching algorithm.

In any case, it would be unrealistic to run any kind of full reasoner –like KAON2 or PowerLoom– on PEIS equipped with poor hardware like embedded devices or motes, so the PEIS-Meta-component will never be turned into a full reasoner. Nothing forbids, instead, to have configuration components provided with full reasoners.

Reasoning for semantic matching does not have to be misunderstood with planning for configuration generation. Reasoning is only the problem of checking admissibility of a configuration, while planning is used as a strategy for searching the configuration space. A configuration component could be featured with both abilities, or none of the two. This last case is the one of the configuration component described in section IV, which is *reactive*, and does not have semantic matching ability at all, being just a prototypical version. There, matching between queries and advertisements is done with a purely syntactic check on the various fields of the component advertisements.

Ontology engineering is the natural field from which to import the tools needed to formally describe the semantics of the PEIS-components. As sketched in section III-A, the domain ontology of the PEIS-Ecology should be referred to in the semantic descriptions of PEIS-components. Any application that has to automatically configure the PEIS-Ecology could refer to this ontology to compute the compatibility between PEIS-components and between the data they should exchange.

Ontologies allow one to implement a sophisticated compatibility mechanism, featuring complex reasoning techniques, since they classify data and objects through a programming language-independent formalism rich enough to express their full *meanings*. PEIS-components could be classified according to their *categories*, according to the *functionalities* they export in the ecology, and according to the *real world objects* to which they relate. The exchanged data could be classified as well through a programming language-independent ontology. For example, if a mobile robot navigation control system needs localization information, the requested data should be a *pose* specifying that it is a planar frame issued by a robot localization system, besides the measurement units and the reference frame.

In the literature, reviews exist that compared various ontology languages from the point of view of expressiveness [14][15]. From the perspective of the end user of an ontology language, other more practical parameters like availability of tools and library support to store/access/process ontology descriptions shall be evaluated. The Knowledge Interchange Format (KIF) and the Web Ontology Language (OWL) are possible candidates as ontology languages for the PEIS-Ecology.

As already pointed out, the configuration application that will be described in section IV features no semantic matching ability, and this extension will be probably the first in our future works.

## IV. ECOLOGY REACTIVE CONFIGURATION

According to the framework described in III-B, special PEIS-components should exist, that are able to generate ecology configurations. These are the `Conf` PEIS-component in the general schema of Figure 3. The implementations of these special components that we present here will be called from now on PEIS-*configurators*. The purpose of PEIS-configurators is that of finding instances of template configurations. The functionality of a PEIS-configurator is indeed fully specified by the template configuration it can instantiate.

The algorithm for reactive configuration implemented into the PEIS-configurators consists in three preliminary phases triggered in sequence after the component is activated. The outcome of the algorithm is the *activation* of the identified configuration or the declaration of the *failure* in accomplishing the configuration task, plus one extra section that continues executing during all the lifespan of the activated configuration. The four phases are listed below.

1) Broadcast queries for discovering instances of the needed,
2) Search for a correct configuration in the space of the instances of the template configuration, given the set of retrieved advertisements,
3) Third-party connection between selected components,
4) Monitoring of the instantiated configuration.

In the following, we explain the different steps of this algorithm. A full example will be provided in the next section.

In phase 1 in order to find components that provide the needed functionalities, the PEIS-configurator issues one query per category of components it is looking for. The PEIS-configurator will organize the received advertisements in different sets, according to the queries to which they refer. Each of these sets represents one of the categories of PEIS-components mentioned in the template configuration. This classification is exploited in phase 2 to cut the space of the searched configurations. Indeed the compatibility of components belonging to categories that are not supposed to be connected according to the template is never checked by the PEIS-configurator.

Within the set of the explored configuration instances, compatibility of PEIS-components is checked referring to their inputs and outputs, as they are listed in the corresponding advertisements. A PEIS-component advertisement is chosen if and only if each of the inputs it lists is compatible with at least one of the outputs of some of the previously chosen components. Input/output compatibility is determined in this prototypical version just by string comparison between the names of the types of inputs and outputs.

The outcome of phase 2 is an instance of a template configuration, i.e. the set of components to be chosen, the parameters to be set, and connections to be created. The mechanisms provided by the PEIS-kernel are exploited in phase 3 to create the connections between the inputs and outputs of the selected components and to set the component

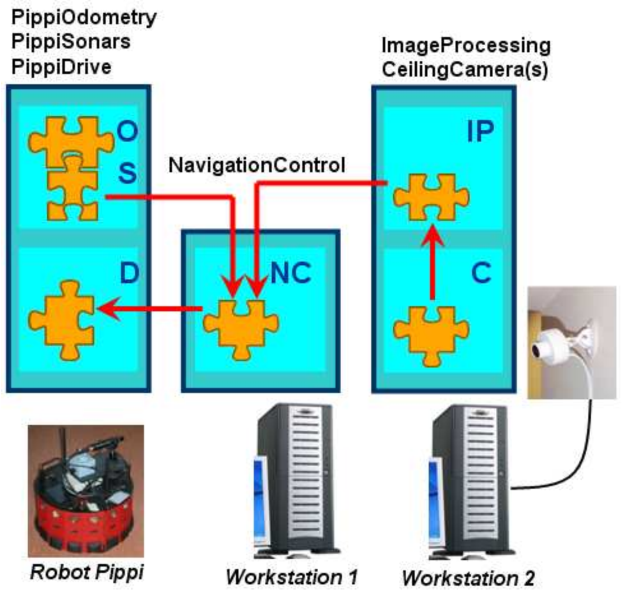


Fig. 7. A sample configuration.

parameters, if needed, while in phase 4 the PEIS-configurator connects itself to a special *failure* signal that can be issued by all PEIS-components. In case a failure signal is received from a component, the PEIS-configurator re-triggers the configuration algorithm.

The capability of reconfiguring the ecology as soon as a failure is detected in the instantiated configuration makes our PEIS-configurators "reactive". PEIS-configurators can reconfigure the ecology when a PEIS-component crashes in a clean way, or when a PEIS-component has to exit the ecology and becomes non available or when a PEIS-component detects that it is not able any more to perform its task, given the occurrence of a non predicted change in the working conditions of the system.

Our PEIS-configurators are classified as "reactive" also because of the locality of the approach they undertake to solve the problem of providing the functionality they declare. PEIS-configurators do not care to find an optimal solution to their problem, but just accept the first correct configuration they can find, and are ready to change it as soon as it fails. No lookahead for possible weaknesses in the long run of the instantiated configuration is performed, nor any coordination strategy is presumed, for optimizing the exploitation of PEIS-components in relation to other parallel tasks of the PEIS-Ecology.

## V. EXAMPLE

In order to fully understand the mechanisms of reactive self-configuration implemented in the PEIS-Ecology, a sample PEIS-configurator will be now illustrated with reference to a concrete run on the PEIS-Ecology test-bed shown in figure 2. The robot used in this experiment, called Pippi, is a Magellan Pro equipped with sonars and odometry.

### A. Experimental setup

Figure 7 illustrates a simple PEIS-Ecology configured for navigating a mobile robot in a dynamic environment. The configuration consists of six PEIS-components and the four connections between them. In this configuration, the navigation control (NC) system takes from the sonars (S) of the robot the occupancy information around the robot, and from an image processing (IP) application the global position of the robot; the navigation control system issues a velocity setpoint to the motor drive (D) of the robot; the image processing application takes one or more video streams from the component interfacing the ceiling camera(s) (C). The image processing and the navigation control PEIS-components need two parameters to be set: respectively the *robot* to track and the *destination* to which to move the robot.

In this PEIS-Ecology two distinct tasks can be identified:

1) the task of *navigating* the robot,
2) the task of *localizing* the robot.

In our example we assume that two distinct PEIS-configurators exist, that can assemble the other PEIS-components and provide the PEIS-Ecology with the above macro-functionalities. These two PEIS-configurators are called respectively the *Navigator* and the *Localizer*.

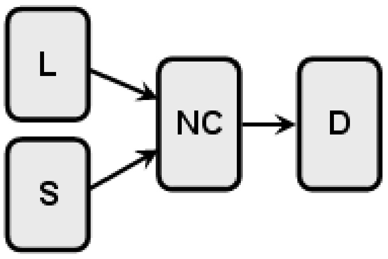


Fig. 8. The configuration template of the Navigator.

Figure 8 shows the configuration template exploited by the Navigator, while Figure 9 shows a possible advertisement for it.

The advertisement of the Navigator is not at all similar to the advertisement of the mobile robot navigation control PEIS-component shown in figure 5. The Navigator does not need any input signal, nor it provides any output signal. It needs instead two input parameters: the name of the robot to move `Robot` and the desired final destination `Place`, and returns one output parameter: a boolean value that should be issued when the robot has reached its destination. All these quantities (`Boolean` included) should be once again instances of classes defined in the domain ontology. The Navigator can be activated in a way that is equivalent to call the *function* whose signature is specified by its advertisement.

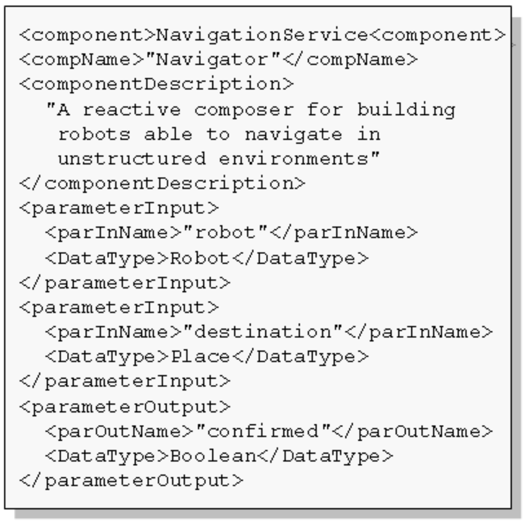
### B. Initial self-configuration

The goal of this experiment is to let the PEIS-Ecology automatically create a configuration like the one depicted in Figure 7. The experiment starts with the Navigator triggered by a call[1] like:

```
navigate(PIPPI, BED)
```

[1]Syntax is simplified allover this example for readability.

$$\begin{tikzcd} \boxed{L} \arrow[dr] & & \\ & \boxed{NC} \arrow[r] & \boxed{D} \\ \boxed{S} \arrow[ur] & & \end{tikzcd}$$



```
<component>NavigationService<component>
<compName>"Navigator"</compName>
<componentDescription>
  "A reactive composer for building
   robots able to navigate in
   unstructured environments"
</componentDescription>
<parameterInput>
  <parInName>"robot"</parInName>
  <DataType>Robot</DataType>
</parameterInput>
<parameterInput>
  <parInName>"destination"</parInName>
  <DataType>Place</DataType>
</parameterInput>
<parameterOutput>
  <parOutName>"confirmed"</parOutName>
  <DataType>Boolean</DataType>
</parameterOutput>
```

Fig. 9. The advertisement of the Navigator.

During phase 1 of the configuration algorithm, the Navigator looks for PEIS-components belonging to the categories stated in its configuration template: a *mobile robot control system*, an *occupancy sensor*, a *localization system*, and a *robot drive*. Even if not shown in Figure 8, also the way the input parameters are treated is specified in the configuration template the Navigator tries to instantiate. More precisely, the Navigator will look for mobile robot control PEIS-components that accept a `Place` as parameter, and for localization PEIS-components that either accept a `Robot` as parameter, or that are *placed* on the `Robot` that is passed as input to the Navigator. The Navigator also queries for an occupancy sensor and a robot drive that reside on the `Robot` that is passed as the input. It does so by broadcasting the following queries:

```
(AND (class~RobotNavigationControl) (canSet~Place))
(AND (class~LocalizationSystem)
        (OR (place=PIPPI) (canSet~Robot)))
(AND (class~OccupancySensor) (place=PIPPI))
(AND (class~RobotDrive) (place=PIPPI))
```

The PEIS-Meta-component installed on Pippi answers to the last three queries sending the advertisements of the PippiOdometry, the PippiSonar, and the PippiDrive. The PEIS-Meta-component installed on Workstation 1 answers the first query with the advertisement of the NavigationControl; the PEIS-Meta-component installed on the PEIS on which also the Localizer is installed answers to the second query with the advertisement of the Localizer. The Navigator receives a total of five advertisements: one per category, except for the localization system, for which it receives two advertisements.

The configuration template states that during phase 2 of the configuration algorithm the Navigator tries to match the outputs of the retrieved mobile robot control system with the inputs of the retrieved robot drive, and the sum of the outputs of the retrieved localization systems and occupancy sensors with the inputs of the retrieved mobile robot control system. It is evident how the template configuration dramatically reduces the number of candidate configurations whose correctness the Navigator tries to check. In this case just two candidate configurations are checked, and, by the way we built our PEIS-components they both succeed.

The three connections that the Navigator generates are:

```
(setTuple :key use-localization-id
          :value (*L* pos.robot) :owner *N*)
(setTuple :key use-sonar-id
          :value (*S* sonar.range) :owner *N*)
(setTuple :key use-setpoint-id
          :value (*N* vel.setvel) :owner *D*)
```

These correspond to calls to the PEIS-kernel API. The Navigator calls the PEIS-kernel also for setting the needed PEIS-components parameters:

```
(setTuple :key "at-me" :value 'BED :owner *N*)
(setTuple :key "track" :value 'PIPPI :owner *L*)
```

Finally the Navigator connects itself to the failure signals from the selected PEIS-components, e.g.:

```
(subscribe :key 'FAIL :owner *L*)
```

Pippi starts to move as soon as the destination is set into the robot navigation control PEIS-component, and it uses the ceiling camera localization, as in the scenario illustrated in Section II.

## C. *Automatic reconfiguration on failure*

A soon as Pippi enters the bedroom -which is out of the field of view of the cameras- the Localizator issues a failure, and the Navigator re-triggers the configuration algorithm. Just one configuration is in the search space, and it is the one that features the PippiOdometry instead of the Localizator for providing the localization of the robot. The three new connections

```
(setTuple :key use-localization-id
          :value (*O* odo.position) :owner *N*)
(setTuple :key use-sonar-id
          :value (*S* sonar.range) :owner *N*)
(setTuple :key use-setpoint-id
          :value (*N* vel.setvel) :owner *D*)
```

are generated, and Pippi can successfully reach the bed, being localized by its odometry.

## VI. DISCUSSION AND CONCLUSIONS

The main contribution of this work is to propose an approach to the dynamic self-configuration of a PEIS-Ecology. Although the approach has been developed and tested with the PEIS-Ecology framework, it is applicable to any distributed robotic system regardless if it has a centralized or decentralized nature. For instance, it should be applicable in ubiquitous robotic systems and network robot systems.

Two challenges in particular make the self-configuration problem especially difficult in these types of systems. First, these systems are highly dynamic: components may be added and removed at all times. Second, they are highly heterogeneous: components may range from static cameras to humanoid robots, and may be built by different developers. We have addressed the first challenge by the mechanism of publishing *advertisements*, and the second one by defining a common, system-wide *ontology*.

These ideas in general, and in particular the semantic interoperability framework described in Section III-B, are inspired both from the JXTA advertisement model [16] and the research on Semantic Web Services (SWS).

In JXTA, however, the matching between advertisements and requests is purely syntactic, although the use of wildcards is allowed. The problem to extend the JXTA framework to allow semantic interoperability is still a research issue, addressed, e.g., in [17].

Research is peculiarly active on the problem of adding semantics to Web Services [18]. OWL was developed specifically for this purpose, and mechanisms for semantic Web Service discovery and composition are rapidly evolving. The state of the art in this field is constituted by the OWL-S ontology and by the frameworks built upon it. We could therefore speculate that one could abstract components of an ecology of robots as Services of OWL-S. Although very attractive in principle, this idea cannot be directly implemented taking the OWL-S Profile ontology as-is. Typical robotic components such as control systems or interfaces to devices are continuous processes that issue data-streams, or require as input a continuous data-stream. By contrast, Web Services are more similar to method invocations: they accept input parameters and issue a return value after a certain time. This fundamental difference is reflected in the difference between the PEIS Profile ontology (Figure 4) and the Service Profile ontology of OWL-S, as discussed in section III-A.

The algorithm for configuration generation that we have presented in Section IV has the typical strength and weakness of any reactive approach: it quickly adapts to the current state of the world, but it might generate non-optimal configurations and it might fail to find a configuration even if one exists. In [19] the use of a global planning approach to generate PEIS-Ecology configurations has been explored instead. Since the overall ontology-based framework presented in this paper can in principle accommodate any configuration algorithm, we could use the plan-based approach in our framework. More interestingly, the plan-based approach and the reactive approach could be combined into a hybrid one: investigating this possibility is one of our next research priorities. Also, the applicability of Web Service composition techniques [20][21] shall be checked, taking into account the conceptual difference between PEIS-components and Web Services that was mentioned above.

## ACKNOWLEDGMENTS

This work has been supported by the Swedish Research Council (Vetenskapsrådet), and by ETRI (Electronics and Telecommunications Research Institute, Korea) through the project "Embedded Component Technology and Standardization for URC (2004-2008)".

## REFERENCES

[1] "Network Robot Forum," www.scat.or.jp/nrf/English/.

[2] T. Sato, T. Harada, and T. Mori, "Environment-type robot system "robotic room" featured by behavior media, behavior content, and behavior adaptation," *IEEE/ASME Transactions on Mechatronics*, vol. 9, no. 3, pp. 529–534, September 2004.

[3] J. Lee and H. Hashimoto, "Intelligent space – concept and contents," *Advanced Robotics*, vol. 16, no. 3, pp. 265–280, 2002.

[4] F. Dressler, "Self-organization in autonomous sensor/actuator networks," in *Proc. of the 19th IEEE Int Conf on Architecture of Computing Systems*, 2006.

[5] J. Kim, Y. Kim, and K. Lee, "The third generation of robotics: Ubiquitous robot," in *Proc of the 2nd Int Conf on Autonomous Robots and Agents (ICARA)*, Palmerston North, New Zealand, 2004.

[6] A. Saffiotti and M. Broxvall, "PEIS ecologies: Ambient intelligence meets autonomous robotics," in *Proc of the Int Conf on Smart Objects and Ambient Intelligence (sOc-EUSAI)*, Grenoble, France, 2005, pp. 275–280.

[7] A. Kameas, I. Bellis, I. Mavrommati, K. Delaney, M. Colley, and A. Pounds-Cornish, "An architecture that treats everyday objects as communicating tangible components." in *PerCom*, 2003, pp. 115–.

[8] J. Rao and X. Su, "A survey of automated web service composition methods," in *Proc of the 1st Int Workshop on Semantic Web Services and Web Process Composition*, San Diego, California, USA, 2004.

[9] G. Tesauro, D. Chess, W. Walsh, R. Das, A. Segal, I. Whalley, J. Kephart, and S. White, "A multi-agent systems approach to autonomic computing," in *Proc of the Int Conf on Autonomous Agents and Multiagent Systems*, 2004, pp. 464–471.

[10] M. Broxvall, M. Gritti, A. Saffiotti, B. Seo, and Y. Cho, "PEIS ecology: Integrating robots into smart environments," in *Proc. of the IEEE Int Conf on Robotics and Automation ICRA*, Orlando, FL, 2006.

[11] "PEIS ecology homepage," www.aass.oru.se/˜peis.

[12] "The DAML Services web site," http://www.daml.org/services/owl-s/.

[13] A. Kaminsky, "Infrastructure for distributed applications in ad hoc networks of small mobile wireless devices," Rochester Institute of Technology, IT Lab, Tech. Rep., may 2001.

[14] O. Corcho, A. Gómez, and Pérez, "Evaluating knowledge representation and reasoning capabilities of ontology specification languages," in *proc. of ECAI-00 Workshop on Applications of Ontologies and Problem-Solving Methods*, Berlin, Germany, 2000.

[15] D. Fensel, "Relating ontology languages and web standards," in *Informatik und Wirtschaftsinformatik. Modellierung*, St. Goar, Germany, 2000.

[16] T. J. community, "The project JXTA web site," www.jxta.org.

[17] D. Elenius and M. Ingmarsson, "Ontology-based service discovery in p2p networks," in *MobiQuitous'04 Workshop Peer-to-Peer Knowledge Management P2PKM 2004*, Boston, MA, USA, aug 2004.

[18] M. Paolucci, T. Kawmura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities," in *Proc. of the First International Semantic Web Conference ISWC*, Sardinia, Italy, jun 2002.

[19] R. Lundh, L. Karlsson, and A. Saffiotti, "Plan-based configuration of an ecology of robots," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Roma, IT, 2007.

[20] D. Berardi, D. Calavese, G. D. Giacomo, M. Lenzerini, and M. Mecella, "Automatic composition of *e-Services* that export their behavior," in *Proc. of the First International Conference of Service Oriented Computing ISOC*, Trento, IT, 2003, pp. 43–58.

[21] P. Traverso and M. Pistore, "Automated composition of semantic web services into executable processes," in *Proc of the 3rd International Semantic Web Conference ISWC*, Hiroshima, JP, 2004, pp. 380–394.