Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Adaptation of System Configuration under the Robot Operating System ⋆

Jonathan M. Aitken * **Sandor M. Veres** * **Mark Judge** *

* *Department of Automatic Control and Systems Engineering, University of Sheffield, UK (e-mail: jonathan.aitken, s.veres, m.judge@sheffield.ac.uk).*

Abstract: This paper lays down the foundations of developing a reconfigurable control system within the Robot Operating System (ROS) for autonomous robots. The essential components of robots are programmed under a ROS system. A formal model is defined as a tripartite graph to represent the robots functional architecture. ROS systems are then generalised to component libraries for any ROS architecture and an abstract model as a "system graph" is introduced. Orthogonality of a library and a system graph is defined and redundancy levels of robot components are studied for maintaining full functionality of the robot by automated reconfiguration in face of hardware malfunction. This allows AI planning tools, such as Planning Domain Definition Language (PDDL), to compute permissible reconfigurations. We present an example of a pair of robotic arms which requires reconfiguration of the underlying control system in order to retain the capability to carry out a task.

Keywords: Autonomous control, Robot arms, Reconfigurable control, Adaptation, Computer-aided control system design

1. INTRODUCTION

The design and deployment of any system is carefully managed by engineers. As systems become more complex (as do their controllers), we gradually wish to incorporate autonomous behaviour into the operation of systems, however, we demand high-performance, high-efficiency and high-tolerance to any failures. The aim of the autonomy should be to simplify the system for the operators whilst, providing an increase in performance or efficiency in the system. However, without careful design and good grounding, the introduction of automation can make the operation of such systems more complex or dangerous - as highlighted in the Überlingen mid air collision. The system operators were unaware of the system configuration, and made a series of incorrect and dangerous decisions based on a lack of knowledge about the operation of the underlying systems (Aitken et al., 2010). This is especially important for any foundation in any mix of autonomous reconfiguration. There is a necessity to be able to provide feedback to the operator to provide better system awareness that is both:

- Accurate, providing a true representation of the current system state and any reconfiguration taking place.
- Understandable, providing a clear and concise representation of the system and reconfiguration. Ideally this should be provided in as simple a language as possible so that the operator can understand the decisions that have been made with the rational behind them in order to develop trust in the automation (Muir, 1987), which is essential to ensuring good operational use, otherwise unexpected autonomous actions could jeopardise trust in the system (Lee and Moray, 1994).

Emerging autonomous system applications are to be implemented in environments where the complexity of what can happen is significantly higher than in traditional feedback control systems, whilst the plant dynamics have relatively low complexity. Automated reconfiguration for these more complex environments is then used in two distinct instances:

- When a new autonomous system is to be created from existing components.
- When an operating autonomous system needs to self-reconfigure due too changes in own hardware or the environment.

This paper sets out a framework for the incorporation of reconfigurable automation in systems design. This framework includes a clear indication of the purpose of reconfiguration which can be used to aid feedback to the operator to provide understanding about the new system configuration. In it we will outline a model of autonomy founded within the Robot Operating System (ROS) (Quigley et al., 2009). This will focus on developing a generic mathematical model of a ROS that is readily analysable by computer.

This generic mathematical model is based upon a tripartite graph, composed of ROS components. By abstracting the ROS systems into libraries of interchangeable elements a model of that can be applied to an robotic system. This "system graph" is a generic model, applicable to any ROS-based system. The focus of reconfiguration is to recover full functionality in the case of component addition (e.g. new

⋆ This work has been supported by the Engineering and Physical Sciences Research Council under grant EP/J011770/1

functionality being added online) or removal (e.g failure). We define the orthogonality of the component library and the system graph, which allows redundancy levels of a ROS system to be identified and so reconfiguration options to be discovered. Using the information a planning system, such as the Planning Domain Definition Language (PDDL), can be used to identify possible reconfiguration options.

Section 2 of the paper discusses literature based around reconfiguration of control systems. Section 3 presents a graph model of the Robot Operating System (ROS). Section 4 provides a methodology of reconfiguration and also gives a formal proof that reconfiguration will be achieved. Section 5 introduces methods based on the symbolic planner in PDDL (planning domain description language) before an example is shown in Section 6. We draw conclusions on the paper in Section 7.

2. RECONFIGURATION

The reconfiguration process is required to change the form of the control system in order to satisfy the changing environment. Ultimately this places a considerable demand on the engineers and system designers and hence automation of the reconfiguration process is a desirable feature to speed up development, eliminate mistakes and obtain a near optimally configured new system. Ideally these control systems should be able to adjust to any situation that they find themselves in, reconfiguring to "accommodate component failures automatically" Zhang and Jiang (2008).

Looze et al. (1985) states four key features that an automatic redesign procedure must address:

- "The failures are unanticipated".
- "The available response time is limited".
- "Non-standard control effectors and configurations may be required".
- "The handling/ride qualities of the reconfigured [system] may be degraded".

2.1 Plug-and-Play

A typical control system will have been designed for the operation of one specific plant. If the characteristics of the plant change at any time, the controller may well cease to operate satisfactorily (Stoustrup, 2009). Additionally new sensors or actuators may become available which make the current control algorithms sub-optimal, traditionally this would require a major redesign in order to incorporate the new components. Plug-and-play control (Ippolito et al., 2005b; Stoustrup, 2009; Bendtsen et al., 2013) focuses on this area of new resources, rather than just dealing with component failure or adaptive control. Plug-and-play control focuses on two distinct challenges (Stoustrup, 2009), first when a known addition is made and system must match an appropriate control scheme. Secondly when an unknown component is plugged in and a pre-existing controller design is not available. In this case awareness of the design is required in order to identify where the new component should be slotted into the control structure, for example using Youla-Kucera-based controlled modification (Bendtsen et al., 2013).

2.2 Reconfigurable and Polymorphic Control

Polymorphic control (Ippolito et al., 2005a; Ippolito and Al-Ali, 2007; Ippolito and Joo, 2008) builds on distributive plug-and-play control. The prime aim of polymorphic control systems is to enhance key system properties, for example performance, resilience and fault-tolerance. This is achieved by modelling the underlying control system as a graph-like structure that is amenable to automatic analysis. The control structure is then part of the dynamic network, as it is fundamentally de-localised amongst the sensors and actuators of the system. Therefore the controller is in a position where it can be quickly reconfigured or restructured in both form and function.

Typically the polymorphic controller is based on a traditional block diagram, for example as built in Simulink, modelled as a graph. Individual components of the system are comprised of multiple vertices that represent the system inputs and outputs. Edges then represent the connection between the input and output vertices. By taking appropriate choices during the modelling process a set of graphs is formed representing the system. These hypergraphs (Papa and Markov, 2006) that are formed can be partitioned to determine new system configurations during operation.

Ippolito and Joo (2008) shows how this polymorphic control can be used to enable collaboration between an Unmanned Ground Vehicle (UGV) and an Unmanned Aerial Vehicle (UAV). In their scenario the UAV is on a approach to landing, when it loses on-board position estimation. A planned (rather than automatic) reconfiguration occurs where a ground based UGV uses vision-based systems to provide position estimation so that a successful (and safe) landing is achieved.

Shore and Bodson (2005) illustrate a reconfigurable control system using a flight of a modified model aircraft capable of four failure modes: frozen elevator, frozen aileron, engine failure and elevator separation. They use realtime recursive identification to find new controller parameters. Rather than adjusting the control structure this process adapts to the failures to counteract failure employing parameter identification in the feedback loops to adjust gains. The new controller models are observed to behave with very good handling characteristics, even under failure.

Agent-based techniques provide a good match to system reconfiguration, as they offer a platform that can easily be distributed. Brennan et al. (2002) uses a collection of distributed agents to reconfigure a distributed control system in real-time. Individual processes are controlled by single agents and an overall coordinator agent is responsible for scheduling tasks. It dispatches mobile agents with tasks to assign to members of the cohort - changing the parameters with which they operate; these mobile agents report back on the success of the tasks execution.

3. THE ROBOT OPERATING SYSTEM

ROS provides a structured communications layer for implementing the software system of a robot that individual processes can use for interaction (Quigley et al., 2009). It simplifies the task of programming robots by providing

a robust framework where the designer is provided with direct control for the robot, without any overhead required in interfacing. A ROS implementation has three typical components:

- *Nodes* - Nodes are the basic process that performs the computation. Each node is a process which may have threads. It is the node where the processing takes place, and that the programmer is responsible for designing. Typical systems are formed from many nodes, each of which does a portion of the overall task.
- *Topics* - Topics are one of the two methods available for exchanging messages. Topics provide a sender-receiver agnostic method for exchanging and receiving messages. In order to publish messages any node can establish a topic and publish messages to it, as and when necessary. Any other node within the network may also publish to this topic. In order receive messages, the other nodes may subscribe, wherein they can receive any message sent via a callback. Therefore a topic provides a many-to-many communication model, as many nodes may publish to a topic (nodes often publish to more than one topic) and many may also subscribe. It is a broadcast messaging stream and so does not provide any synchronous message transfer.
- *Services* - Services provide a more strict communication model where there is an established request and response message. In a process similar to web services, a node may request certain information via a service and then be supplied back with the information on demand. This contains much lower overheads than a topic, as there is no continual broadcast of messages rather an individual exchange.

Communication between these three components is facilitated by *Messages* which are a bundle of information packaged as a strictly typed data structure. Typically these are comprised of standard primitive types, but also may contain other messages.

There are two possibilities for describing a ROS based system:

- A tri-partite graph with vertices for nodes, topics and services.
- Nodes with labelled, directed edges for topics and services.

The second representation is less amenable to exact mathematical analysis. Hence our choice remains the first option, the tri-partite graph, where each of the three vertex types are not interchangeable in graph matching algorithms. New topics and services can be easily introduced that can allow reconfiguration of the system to provide agents with the information they required, albeit sourced from different locations. This can be seen in Figure 1, where vertices represent nodes, topics and services. Edges show the routes of information flow, all services and topics therefore must have at least one incoming edge from a node. Additionally all node communication must occur through topics or services.

Definition 1. A ROS-graph is $G_{ROS} = \{N, T, S, E\}$, where N are the set of vertices of nodes, T are the set of topics and S are the set of services. E represents a set of directed edges between vertices in N and T or N and S in either direction. A labelling function λ is a mapping defined over all vertices as follows: $\ell : T \cup S \rightarrow \Omega$ and $\aleph : E \rightarrow \Gamma$ where Γ is the class set of an agent ontology which define templates for message objects.

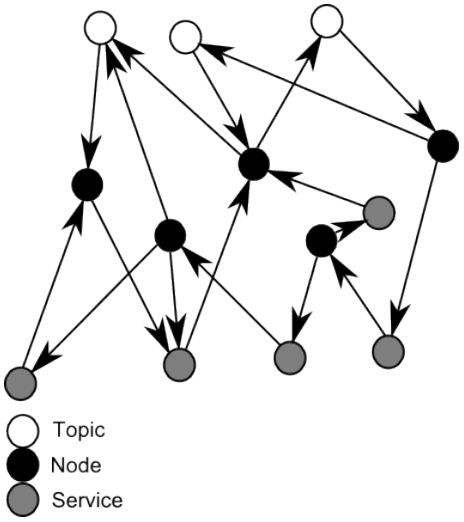


Fig. 1. Basic ROS-Graph.

To serve reconfigurability we will make a distinction between a ROS-graph and a ROS system. A ROS-graph is as defined above. A ROS system however is a set of nodes each of which is potentially able to provide a number of services and publish a set of topics. Whether these will be used depends on whether other nodes use these services or subscribe to these topics. Some of the services of topics can remain dormant if no other node uses them. Hence we will make a distinction between a ROS-graph and the collective potential of a set of nodes in terms of the services and topics they can provide. We will call this a *ROS system.*

Definition 2. A *ROS system provision* $R = [N, S^+, T^+]$ is a set of nodes N with their available services $S^+ : N \rightarrow 2^{(\Omega \times \Gamma)}$ and available publishing capabilities $T^+ : N \rightarrow 2^{(\Omega \times \Gamma)}$ through topics.

Note that this definition does not exclude the possibility that certain services can be provided by more than one nodes or that identically labelled and formatted messages could be published by several nodes.

A concrete ROS-graph is however not defined by node capabilities but by the needs of nodes for some services and topics.

Definition 3. ROS system requirement $[N, S^-, T^-]$ is a set of nodes with required services $S^- : N \rightarrow 2^{(\Omega \times \Gamma)}$ and topics $T^- : N \rightarrow 2^{(\Omega \times \Gamma)}$.

As there can be multiple providers, the ROS system requirement will not uniquely define a ROS-graph either as different graphs can be obtained by different choices of providers.

Lemma 4. For any ROS-graph $G = \{N, T, S, E\}$ the relations $\bigcup_{n \in N} S_G^-(n) = S_G^+ \subseteq S_G \subseteq S_R^+$ and $\bigcup_{n \in N} T_G^-(n) = T_G^+ \subseteq T_G \subseteq T_R^+$ hold where the definitions $S_G^+ = \bigcup_{n \in N} S_G^+(n)$ and $T_G^+ = \bigcup_{n \in N} T_G^+(n)$ are used.

This lemma means that within a ROS-graph the set of service requirements and topic publications are satisfied. This sheds light on the fact that a ROS-graph is only a single realisation of the potentials of ROS system provision in R. The set of all ROS-graphs which can be obtained under a provision R will be denoted by $\Psi(R)$. If we assume that multiples of the same nodes can be used from R then $\Psi(R)$ is an infinite set.

One can think of a node set N used above not only as the vertices of a ROS-graph with fixed services and topics but as a library of nodes which can be configured to form an autonomous system. A *ROS system provision* $R = [N, S^+, T^+]$ can be thought of not as the set of nodes to be used in a system but as a library of available resources to synthesize an autonomous control capability. The next section will examine how reconfiguration can be done in engineering terms.

4. MAIN RESULTS ON RECONFIGURATION

This section provides a methodology of reconfiguration and also gives a formal proof that reconfiguration will be achieved. Figure 2 displays the structure of a typical control system as part of an autonomous system.

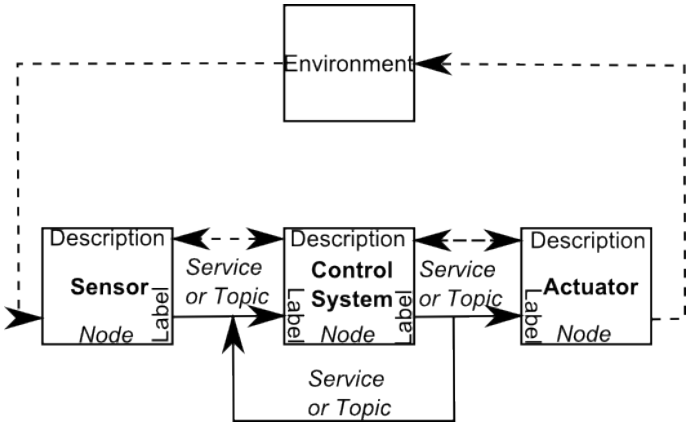


Fig. 2. Control Block Diagram Indicating ROS-graph Components.

A key question is: how will the software agent (i.e. the agent handling the reconfiguration) able to decide whether a feedback control system as on Figure 2 will satisfy quality and performance requirements? To state the question more precisely: assuming that the agent is able to find a *correct* connection of components (i.e. sensor nodes, sensor services, perception nodes and services, controller nodes and services, actuator nodes and services) how can we make sure that the resulting control system will serve the purpose? Before we can answer this important question a constructive analysis and some observations can be made.

(1) If perception nodes (i.e. sensor signal processing nodes) and control nodes are not adaptive to varying quality of sensor signals and actuator qualities then system reconfigurability becomes much more hard. It is therefore desirable, and in fact a practical necessity that in a reconfigurable ROS architecture the algorithmic solutions to sensor signal processing and control are adaptive: adaptive signal processing and adaptive control is a necessity. Adaptivity means that substitution of sensors and actuators with a bit different qualities will not cause the control system to fail completely in terms if instability or bad performance.
(2) Assuming sufficient level of adaptivity is ensured in perception nodes and control nodes, the question remains: how to decide which components can be connected, how to derive and choose components which can tolerate each other due to adaptivity?

We outline two types of solutions to this question: one takes a longer time and the other can be used quickly and during the robots operation. Both assume that some configuration Σ of sensors, perception processing, adaptive/learning control and actuators have been tested by the agent for correct data connectivity in terms of services and topics.

M — The *M*-solution assumes that there are dynamical models available for each of the components of Σ. The agent is also assumed to be equipped with the ability to simulate each of the components and typical environmental model to be able to generate simulated sensor signals and effects of actuators. The agent performs a set of simulations and evaluates the control performance results. This can be a lengthy process and mostly applicable when engineers need to set up a new system. A remotely stationed autonomous robot, without communication with humans supervisors, may however also have the time to evaluate its reconfiguration options. This solution assumes no prior data are available about the quality of performance of Σ.

S — The *S*-solution relies on *a priori* stored configuration library L of *interoperability schemas* which is a list of system graphs with abstracted node names and direction of connectivity indicated by edges. For quick reconfiguration the agent needs to check whether there is a ROS-graph which matches a required interoperability schema and can be implemented using the resources in the library L.

In the following we provide a formal representation of a system for the *S*-solution, which will be our main interest in the rest of the paper.

Definition 5. (1) An *abstraction* of the node set of a *ROS system provision* library $L = [N, S^+, T^+]$ is a mapping $N \to \Lambda$ of nodes to a label set Λ.
(2) A *system graph* is a directed graph with vertices labelled from Λ.

To clarify the relevance of these abstract definitions we present an example before we proceed to our main result.

Example 6. Assume that we have the following library of components and associated ROS nodes: (1) a five joint DC-motor-based robot arm with actuator node N_{DC} (2) a five joint servo-motor-based robot arm with actuator node N_{SM} (3) a five servo motor and 2 camera based vision head with actuator node N_{VH} (4) a GPGPU cluster based perception module represented by ROS node N_{VM} (5) a vision based adaptive feedback control module represented by node N_{FB} (6) a human typed-command based task interpreter module represented by ROS-node N_{TEX} (7) a human-voice control based task interpreter module represented by ROS-node N_{VOI} . Each of these nodes provide services and require some services to be

Node	Λ
N_{DC}	robot_arm_actuator
N_{SM}	robot_arm_actuator
N_{VH}	robot_vision_head
N_{VM}	vision module
N_{FB}	feedback control module
N_{TEX}	task interpreter
N_{VOI}	task interpreter

Fig. 3. Example abstractions of ROS nodes.

available. A mapping into Λ is illustrated in Figure 3 where two pairs of nodes map to the same abstraction labels. A system graph is illustrated in Figure 4. The automated

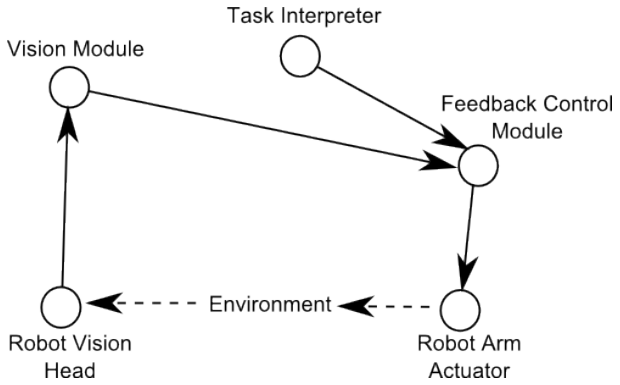


Fig. 4. A system graph to carry out tasks of the robot arm. The dashed arcs are not part of the directed edges of the system graph, they are used to indicate the role of the physical environment of the robot arm.

reconfiguration problem is now to find services or topics to realise the functionality of the system graph based on the resource from library L.

We define a library of control schemas to be more then just a set of system graphs: we associate a skill and environment description with each graph.

Definition 7. An *interoperability schema* S is a triple $S = [G, x, e]$ where G is a system graph and x is a textual (qualitative) description of the skill of the robot which the graph represents and e is a textual (qualitative) description of the environmental dynamics in which the skill can operate.

Note that here skill can represent a perception/modelling skill, physical/control skill or mental/planning skill of the autonomous system/robot. In this paper we leave the method of textual description of x and e as a free design variable. As an example we quote the use of sEnglish (Veres, 2012, 2008, 2010) for a description of the skill or the environment.

Definition 8. A *configuration library* C is defined as a list of interoperability schemas. The *component library* L is a set of available sensor, perception and control nodes with a set of service and publishing capabilities each. The C is called *independent* (or *orthogonal*) to L if any substitution of correctly labelled components into any interoperability schema results in the possibility of a correctly functional control system.

Our main result is that for autonomous system design ensuring the orthogonality of L to C is a worthwhile a priori undertaking as it results in considerable time savings when autonomous systems are reconfigured in practical engineering operations.

Theorem 9. Assuming C is orthogonal to L the number of implementable ROS-graphs is multiple of the number of system graphs.

Proof. If there is only one ROS node available for each node of a system graph then there is no redundancy of components. Still, the way services or topics are set up to represent directed edges of the system graph, can be in multiple ways. In case of multiple ROS nodes matching a system node, orthogonality of C to L means that there will be service/topic solutions fore each of the substitutions of abstractions by available nodes. □

Orthogonality of C to L means that system schemas can be arbitrarily "plugged in" by matching name nodes as service/topic communication (represented by edges of the system graph) can always be constructed by the configuration agent to obtain a functioning ROS graph for implementation. The greater the reconfigurability in face of unavailable nodes the greater the power of this approach. Here we present a quantitative analysis in case a large number of components are needed to build up an autonomous system or robot.

Assume that the probability of the availability of a library component $c \in L$, which can rely on a piece of hardware, is denoted by $p(c)$. The abstraction of the components in L will be denoted by $\nu : L \to \Lambda$. We also assume, without imposing real limitations, that C requires the use of all of $\nu(L) = \Lambda$. We are interested to obtain the probability of the autonomous system C not being disabled by lack of reconfigurability.

If we assume that different $a \in \Lambda$ can be played by an uneven number of library components $card(\nu^{-1}(a))$ then [1] the complexity of computations goes up and we do not get a clear picture about the redundancy level present. For the sake of investigating the relationship between the number of components in C and the corresponding redundancy level to maintain a certain probability level of reconfigurability, we assume that $r = card(\nu^{-1}(a)),\ a \in \Lambda$ where r now means our level of redundancy.

Theorem 10. For a given probability $q \approx 1$ of reconfigurability we can state the following:

(1) If q is required and $N \to \infty$ then the redundancy level r needed is proportional to $\log N$.

(2) If N is fixed and $q \to 1$ then the required redundancy is asymptotically proportional to $\log\log q^{-1}$.

(3) Otherwise an approximate formula for the redundancy level to ensure reconfigurability with probability q is

$$r = \frac{\log(-\log q)}{\log p} - \frac{\log N}{\log p}$$

[1] Here $card(x)$ denotes the number of elements in set x.

Proof. The essence of the proof is based on discovering what happens to r to maintain the relationship

$$(1 - p^r)^N = q$$

This leads to

$$r = \frac{\log\left(1 - e^{\log q/N}\right)}{\log p}$$

Given that $\log q/N \approx 0$ the approximation $\exp\{\log q/N\} \approx 1 + \log q/N$ can be made that results (3) as

$$r = \frac{\log(-\log q)}{\log p} - \frac{\log N}{\log p}$$

which also implies (1) and (2) when $N \to \infty$ or $q \to 1$. □

If control system reconfiguration is needed due to changing resources (either during robot operations or at design stage) the process of finding available components for the system graph can be automated. The next section illustrates this automated procedure using PDDL.

5. SOLVING RECONFIGURATION BY PDDL

So far this paper has developed a framework that provides a mathematical model for describing ROS systems that allows for autonomous reconfiguration. However, this has stopped one step short of how to handle this information to allow autonomous reconfiguration. We have presented a representation of the system as a ROS-graph. We will show that reconfiguration of this graph, and thus preservation of the system properties, can be accomplished by using appropriate planning constructs described by the Planning Domain Definition Language (PDDL) (McDermott et al., 1998; Mcdermott, 2000; Fox and Long, 2003).

PDDL is the defacto standard for describing planning problems (McDermott et al., 1998; Mcdermott, 2000). It uses STRIPS actions supported by ADL conditional effects. PDDL describes what is present in a problem: what predicates there are, the actions and their effects, and the structure of any compound actions. The language is divided into sets of features (requirements) with each problem domain specifying which requirements are needed. The most commonly used include STRIPS, ADL, equality, and typing. In order to fully describe a problem, two files are generally used. The first, the domain file, contains the domain definition and the complete set of action schemata, defined by required preconditions and associated effects. The second file, the problem file, contains a listing of the objects used for a specific problem instance in this domain.

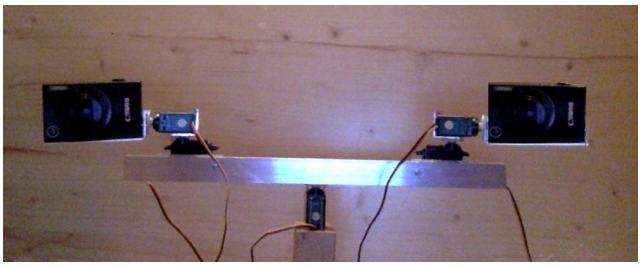
This PDDL domain can be used to describe a simple reconfiguration over the ROS-graph defined in Section 3. The effect of each reconfiguration will have an impact on the system behaviour and ultimately determine the effectiveness of the overall system. The planner is free to develop a new system configuration, the effectiveness of this controller configuration can then be assessed using traditional control techniques to assess the result of these planning steps, such as steady state error and settling time.

In order to begin defining a reconfigurable ROS-graph we must define an appropriate domain structure that can represent the reconfigurable control system. This implies that the ROS-graph must be broken down in order to fully model the components. The ROS-graph contains services, topics and nodes. Services and topics are the medium for exchanging messages. The nodes represent the core components of the control system, which includes sensors, control processes and actuators. Sensors take information from the environment (e.g. measurements of actuator positions or measurable system states). Control processes manipulate information, with the resulting signals flowing to actuators or to further control systems. Actuators receive information from control systems to effect changes on the environment.

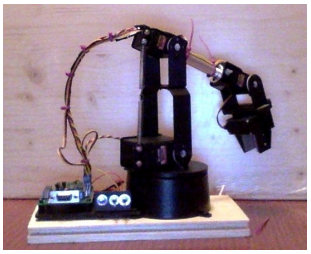
Figure 2 shows a conceptual block diagram of the ROS-graph realised reconfigurable control system. Sensors, actuators and control blocks are formed from ROS nodes, which use services or topics to communicate. Under reconfiguration there are two components that exist for the planner to ensure that connections can be made: labels associated with the data provide a useful matching tool to ensure data consistency, and block matching block descriptions provide further definition.

6. AN EXAMPLE OF RECONFIGURATION

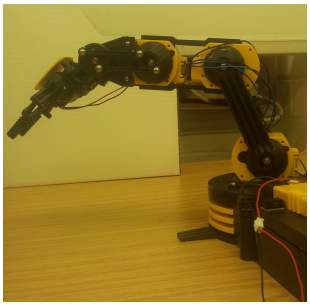
In order to explore the reconfiguration of a ROS-graph, we will introduce an example that provides several opportunities for reconfiguration. A robotic arm is required to perform two separate tasks, "Object Repositioning" and "Disc Loading". The configuration of the system is summarised in Table 1. In order to perform the task there are two arms available; the first is a five-degree-of-freedom (5-dof) jointed robotic arm (shown in Figure 5(b)) with digital servos which provides accurate information on their position, the second is a 5-dof jointed robotic arm which is controlled only by DC motors (shown in Figure 5(c)).



(a) Camera Setup.



(b) Digital Servo Controlled Robotic Arm.



(c) DC Motor Controlled Robotic Arm.

Fig. 5. Robotic System Components.

The components in the system can be broken down into ten broad classes:

Table 1. Robot Example Components, Services and Schema Assignment

Node	Services	Services Requests	In Schema
NDC1 NDC2 NDC3 NDC4 NDC5	**Request Class:** voltage pulse, **Service:** dc servo command	none	Object repositioning
NSM1 NSM2 NSM3 NSM4 NSM5	**Request Class:** digital signal, **Service:** digital servo command	none	Object repositioning, Disc loading
NCA1 NCA2	**Request Class:** camera setting **Response Class:** rgb640x480, **Service:** camera image	none	Object repositioning, Disc loading
NCO1 NCO2	**Request Class:** Boolean **Response Class:** pressure vector, **Service:** gripper feeling	none	Object repositioning, Disc loading
NVP1 NVP2	**Request Class:** rgb640x480, pressure vector, arm features, object features **Response Class:** relative position (NCO1), description (NCO1), metric positions model (NCO2) **Service:** visual perception	camera image, gripper feeling	Object repositioning, Disc loading
NFB11 NFB12 NFB13 NFB14	**Request Class:** object features, arm features, **Response Classes:** Completion Statement **Service:** positioning arm (NFB11), grasping object (NFB12), carrying object (NFB13), placing object (NFB14)	gripper feeling, visual perception, dc servo command	Object repositioning
NFB21 NFB22 NFB23 NFB24	**Request Class:** object features, arm features, **Response Classes:** Completion Statement **Service:** positioning arm (NFB11), grasping object (NFB12), carrying object (NFB13), placing object (NFB14)	gripper feeling, visual perception, digital servo command	Object repositioning, Disc loading
NREA	**Request Class:** human request, object features, **Response classes:** Boolean, sentence	visual perception, gripper feeling, dc servo command digital servo command	Object repositioning, Disc Loading

- NDC1, NDC2, NDC3, NDC4, NDC5 - These are control components for the DC motor controlled arm, each controller serves one joint. These are only capable of being used for the Object repositioning task. These control the base, shoulder, elbow, wrist and gripper respectively.
- NSM1, NSM2, NSM3, NSM4, NSM5 - These are control components for the digital servo controlled arm, each controller serves one joint. These are capable of being used for the Object repositioning or the Disc loading task. These control the base, shoulder, elbow, wrist and gripper respectively.
- NCA1 and NCA2 - These are cameras fitted that are available to provide images of the scene. These are both capable of being used for either the Object repositioning or Disk loading task.
- NCO1 and NCO2 - These are pressure sensors fitted to the object grippers. Both can be used for Object repositioning and Disc loading.
- NVP1 and NVP2 - These are computer vision algorithms that are capable of interpreting an incoming image and providing more information about it. NVP1 provides a relative position and description of the work piece and can be used for Object repositioning and Disc loading. NVP2 provides different information about the work piece, as it produces a metric positions model which can be used for either Object repositioning or Disc loading.
- NFB11 and NFB21 - These are control tasks that position the arm, NFB11 only operates with the DC motor controlled arm so is only available for Object repositioning. NFB21 only operates the digital servo controlled arm so is available for both the Object repositioning and Disc loading tasks.
- NFB12 and NFB22 - These are control tasks that grasp objects, NFB12 only operates with the DC motor controlled arm so is only available for Object repositioning. NFB22 only operates the digital servo controlled arm so is available for both the Object repositioning and Disc loading tasks.
- NFB13 and NFB23 - These are control tasks that carry objects, NFB13 only operates with the DC motor controlled arm so is only available for Object repositioning. NFB23 only operates the digital servo controlled arm so is available for both the Object repositioning and Disc loading tasks.
- NFB14 and NFB24 - These are control tasks that places objects, NFB14 only operates with the DC motor controlled arm so is only available for Object repositioning. NFB24 only operates the digital servo controlled arm so is available for both the Object repositioning and Disc loading tasks.
- NREA - This is the reasoning control node that coordinates the action.

The planner performs reconfiguration on the system, creating a new system using different components that is still capable of completing the assigned task. However, there is a generic model that sits underneath that the planner is following. In general this follows the form shown in Figure 2. The general form of the controller is shown in Figure 6. There are a set of sensor nodes that are capable of measuring the presence of an object within the gripper and cameras capable of producing an image of the environment which can be processed by a perception unit. Control systems carry out the four basic actions of the robot arm: positioning the arm, carrying, grasping and placing an object, by interacting with the five joints within the robot arm.

Under reconfiguration the PDDL planner selects appropriate components from Table 1 to place in each of the boxes in the diagram. In order for the blocks to be compatible, the services that each block requires must be matched. Therefore there are a finite set of components that can be used together. NDC1, NDC2, NDC3, NDC4 and NDC5

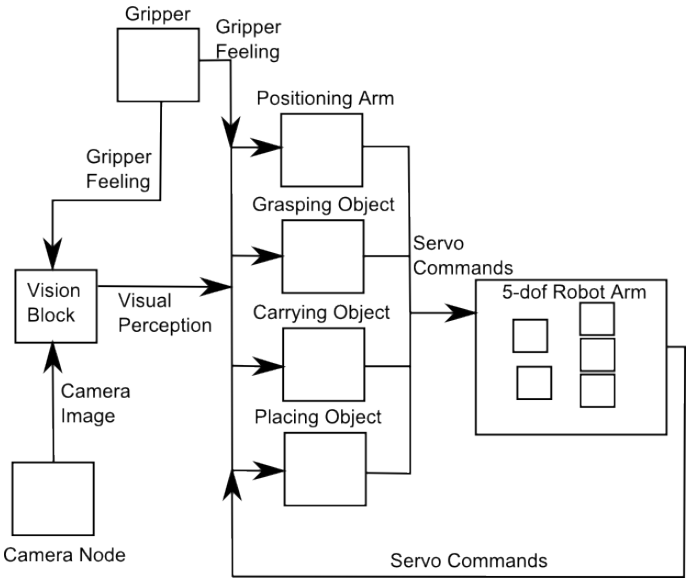


Fig. 6. Basic Block Diagram for the Reconfigurable Robot-Arm.

all related to the DC motor controlled arm, therefore they must be used with NFB11, NFB12, NFB13, NFB14 and NFB15 which are control nodes for DC servos. Similarly NSM1, NSM2, NSM3, NSM4 and NSM5 are the actuators for the servo controlled arm so may only be used with NFB21, NFB22, NFB23, NFB24 and NFB25, this is encoded into the services produced and consumed. By ensuring the labels match ("digital servo command" or "dc servo command"), only appropriate controllers may be connected.

Additionally there are a set of physical constraints placed on the system, the camera, gripper and vision perception units are assumed to be linked together, therefore selection of NVP1 implies NCO1 and then NCA1 must be used. Selection of NVP2 implies that NCO2 and then NCA2 must be used as part of the complete system.

Taking these restrictions as a whole it becomes clear that there are several constraints in the problem, before considering the reconfiguration. Namely either robot arm can be used to perform the "Object Repositioning" task, whereas the "Disc Loading" task can only be performed by the robot with the digital servo motors. Overall these design constraints allow for four possible configurations in "Object Repositioning" and two in "Disc Loading".

Within "Object Repositioning", these four cases arises as either robot arm may perform the task, but there is freedom as to whether Camera 1 (NCA1) or Camera 2 (NCA2) (as shown in Figure 5(a)) is used to provide images to the perception unit. This provides four possible configurations:

- NDC1, NDC2, NDC3, NDC4, NDC5, NCA1, NCO1, NPV1, NFB11, NFB12, NFB13, NFB14, NFB15, NREA
- NDC1, NDC2, NDC3, NDC4, NDC5, NCA2, NCO2, NPV2, NFB11, NFB12, NFB13, NFB14, NFB15, NREA

```
(:action visual_perception_ready
    :parameters  ;available to any schema
        (?visualperceptnode1 - visualperceptnode
        ?grippernode1 - grippernode
        ?cameranode1 - cameranode)
    :precondition
        (and (config_vispercept ?visualperceptnode1 ?grippernode1 ?cameranode1)
            (image_available ?cameranode1)
            (gripper_sensing_available ?grippernode1)
            (no_visual_percept_available ?visualperceptnode1))
    :effect
        (and (visual_percept_available ?visualperceptnode1)
        (not(no_visual_percept_available ?visualperceptnode1))))
```

Fig. 7. Example Action Block for Visual Perception.

- NSM1, NSM2, NSM3, NSM4, NSM5, NCA1, NCO1, NPV1, NFB21, NFB22, NFB23, NFB24, NFB25, NREA
- NSM1, NSM2, NSM3, NSM4, NSM5, NCA2, NCO2, NPV2, NFB21, NFB22, NFB23, NFB24, NFB25, NREA

However, the "Disc Loading" task is more complex and so can only be performed by one robot which presents two possible solutions:

- NSM1, NSM2, NSM3, NSM4, NSM5, NCA1, NCO1, NPV1, NFB21, NFB22, NFB23, NFB24, NFB25, NREA
- NSM1, NSM2, NSM3, NSM4, NSM5, NCA2, NCO2, NPV2, NFB21, NFB22, NFB23, NFB24, NFB25, NREA

6.1 Implementing the System in PDDL

In order to implement the system in PDDL we require two separate files, one which describes the domain and one for the problem description.

Definition 11. A *PDDL problem domain* can be described $D = \{Req, Ty, Pred, A\}$, where Req are the requirements defining the object types. Ty are the possible types defined in the problem, where $T = \{Objs, Sch\}$ is a collection of objects, $Objs$ and schemas, Sch. These objects can be further defined $Objs = \{RObjs\}$ which contain a set of ROS objects ($RObjs$) - where $RObjs = \{N\}$ is a collection ROS nodes N. $Pred$ contains the predicates which describe whether the basic connection conditions have been met for each of the nodes and a Messaging Type ($Mtyp$) that form the subcomponents, $Pred = \{N, Mtyp\}$, where $Mtyp$ is either a Service or Topic, $Mtyp = \{T, S\}$. The Actions A are defined as containing, parameters taking part in the action $Para$, preconditions (Pre) and effects (E), $A = \{Para, Pre, E\}$. Where parameters, $Para = \{N\}$, is a set of ROS nodes, the preconditions, $Pre = \{Pred^{pre}\}$, are a set of predicates that hold before the action and the effects, $E = \{Pred^{efx}\}$, are predicates that hold after the action takes place.

Actions control how each of the components can be connected. Figure 7 shows an example PDDL action for visual perception, showing required preconditions and effects. That is, to set up the perception block, it must not previously be established, it must use an image from a camera, and a gripper must be available. Each of the components that form part of the overall system must be described, using the appropriate labels and descriptions found within the ROS-graph detailing connection constraints.

```
Object Repositioning Config plan 1:
        0: DC_SERVO_BASE_TO_SHOULDER DCBASENODE1 DCSHOULDERNODE1 OBJECTREPO1
        1: DC_SERVO_SHOULDER_TO_ELBOW DCBASENODE1 DCSHOULDERNODE1 DCELBOWNODE1
        2: DC_SERVO_ELBOW_TO_WRIST DCSHOULDERNODE1 DCELBOWNODE1 DCWRISTNODE1
        3: DC_SERVO_WRIST_TO_GRIPPER DCELBOWNODE1 DCWRISTNODE1 DCGRIPPERNODE1
        4: SETUP_CAMERA_IMAGE CAMERANODE2
        5: DC_SERVO_AVAILABLE DCWRISTNODE1 DCGRIPPERNODE1 DCSERVO1
        6: SETUP_GRIPPER_SENSING GRIPPERNODE2
        7: VISUAL_PERCEPTION_READY VISUALPERCEPTNODE2 GRIPPERNODE2
                CAMERANODE2 POSITIONARMNODE2
        8: DC_PLACE_OBJECT DCSERVO1 GRIPPERNODE2 VISUALPERCEPTNODE2
                POSITIONARMNODE2 DCPLACINGOBJECTNODE1

Object Repositioning Config plan 2:
        0: DIGITAL_SERVO_REPOSITION_VERSION_BASE_TO_SHOULDER DIGITALBASENODE1
                DIGITALSHOULDERNODE1 OBJECTREPO1
        1: DIGITAL_SERVO_REPOSITION_VERSION_SHOULDER_TO_ELBOW DIGITALBASENODE1
                DIGITALSHOULDERNODE1 DIGITALELBOWNODE1
        2: DIGITAL_SERVO_REPOSITION_VERSION_ELBOW_TO_WRIST DIGITALSHOULDERNODE1
                DIGITALELBOWNODE1 DIGITALWRISTNODE1
        3: DIGITAL_SERVO_REPOSITION_VERSION_WRIST_TO_GRIPPER DIGITALELBOWNODE1
                DIGITALWRISTNODE1 DIGITALGRIPPERNODE1
        4: SETUP_CAMERA_IMAGE CAMERANODE1
        5: DIGITAL_SERVO_REPOSITION_VERSION_AVAILABLE DIGITALWRISTNODE1
                DIGITALGRIPPERNODE1 DIGITALSERVO1
        6: SETUP_GRIPPER_SENSING GRIPPERNODE1
        7: VISUAL_PERCEPTION_READY VISUALPERCEPTNODE1 GRIPPERNODE1 CAMERANODE1
                POSITIONARMNODE2
        8: DIGITAL_PLACE_OBJECT DIGITALSERVO1 GRIPPERNODE1 VISUALPERCEPTNODE1
                POSITIONARMNODE2 DIGITALPLACINGOBJECTNODE1
```

Fig. 8. Planner Output.

6.2 Results of PDDL Reconfiguration

We have used the FF planner (Hoffmann and Nebel, 2001) to solve an instance of the reconfiguration problem. We provide in Figure 8 the plan steps generated by the planner. Within 18 steps the planner successfully produces a configuration that satisfies the "Object repositioning" task for both DC and servo controlled arms.

7. CONCLUSION

This paper lays a foundation by defining a reconfigurable control system within the ROS. This allows the development of autonomously reconfigurable robot systems. A formal model has been developed using a tripartite graph to represent a ROS implementation of a robotic system. We have abstracted ROS components into a library. This allows a system graph to be developed and implemented in a planning language such as PDDL. A planner is then capable of producing reconfigurations of the underlying system that satisfy a given task. Our results, using the FF planner, demonstrate the efficacy of this approach. Future work includes augmenting our PDDL prototype with functions and durative actions. This will allow for a number of separate metrics (e.g. steady-state error and settling time), facilitating multiple-criteria optimisation.

REFERENCES

Aitken, J., Alexander, R., and Kelly, T. (2010). A Case for Dynamic Risk Assessment in NEC Systems of Systems. In *5th International Conference on System of Systems Engineering (SoSE)*.

Bendtsen, J., Trangbaek, K., and Stoustrup, J. (2013). Plug-and-Play Control Modifying Control Systems Online. *IEEE Transactions on Control Systems Technology*, 21(1), 79–93.

Brennan, R., Fletcher, M., and Norrie, D. (2002). An Agent-Based Approach to Reconfiguration of Real-Time Distributed Control Systems. *IEEE Transactions on Robotics and Automation*, 18(4), 444–451.

Fox, M. and Long, D. (2003). PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.(JAIR)*, 20, 61–124.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14, 253–302.

Ippolito, C. and Al-Ali, K. (2007). Topological Constructs for Automatic Reconfiguration of Polymorphic Control Systems. In *AIAA Infotech@ Aerospace 2007 Conference*.

Ippolito, C., Al-Ali, K., and Dolan, J. (2005a). Polymorphic Control and Trajectory Optimization of an Autonomous Ground Vehicle over Wireless Mobile Networks. In *AIAA Infotech@ Aerospace 2005 Conference*.

Ippolito, C. and Joo, S. (2008). Polymorphic Control Reconfiguration in an Autonomous UAV with UGV Collaboration. In *IEEE Aerospace Conference*.

Ippolito, C., Pisanich, G., and Al-Ali, K. (2005b). Component-Based Plug-and-Play Methodologies for Rapid Embedded Technology Development. In *AIAA Infotech@ Aerospace 2005 Conference*.

Lee, J. and Moray, N. (1994). Trust, Self-Confidence, and Operators' Adaptation to Automation. *International Journal of Human-Computer Studies*, 40(1), 153–184.

Looze, D., Weiss, J., Eterno, J., and Barrett, N. (1985). An Automatic Redesign Approach for Restructurable Control Systems. *IEEE Control Systems Magazine*, 5(2), 16–22.

McDermott, D., Ghallab, M., Howe, A., and Knoblock, C. (1998). PDDL-The Planning Domain Definition Language. Technical report, New Haven, CT: Yale Center for Computational Vision and Control.

Mcdermott, D. (2000). The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2), 35–56.

Muir, B. (1987). Trust Between Humans and Machines, and the Design of Decision Aids. *International Journal of Man-Machine Studies*, 27(5-6), 527–539.

Papa, D. and Markov, I. (2006). Hypergraph Partitioning and Clustering. *Approximation Algorithms and Metaheuristics*, 1–38.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). ROS: An Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Shore, D. and Bodson, M. (2005). Flight Testing of a Reconfigurable Control System on an Unmanned Aircraft. *Journal of Guidance, Control, and Dynamics*, 28(4), 3747–3752.

Stoustrup, J. (2009). Plug & Play Control: Control Technology Towards New Challenges. *European Journal of Control*, 15(3-4), 311–330.

Veres, S.M. (2010). Theoretical foundations of natural language programming and publishing for intelligent agents and robots. In *TAROS 2010, Towards Autonomous Robotic Systems, UK Conference*. Plymouth, UK.

Veres, S.M. (2012). Knowledge of machines: review and forward look. *J. of Systems and Control Engineering*, 226(1), 3–10.

Veres, S. (2008). *Natural Language Programming of Agents and Robotic Devices*. SysBrain, London.

Zhang, Y. and Jiang, J. (2008). Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2), 229–252.