Service Oriented Robotic Architecture for Space Robotics: Design, Testing, and Lessons Learned

Lorenzo Flückiger
Carnegie Mellon University
NASA Ames Research Center, Mail Stop 269-3
Moffett Field, CA-94035, USA
Lorenzo.Fluckiger@nasa.gov

Hans Utz
Carnegie Mellon University
NASA Ames Research Center, Mail Stop 269-3
Moffett Field, CA-94035, USA
Hans.Utz@nasa.gov

Abstract

This paper presents the lessons learned from six years of experiments with planetary rover prototypes running the Service Oriented Robotic Architecture (SORA) developed by the Intelligent Robotics Group (IRG) at the NASA Ames Research Center. SORA relies on proven software engineering methods and technologies applied to space robotics. Based on a Service Oriented Architecture and robust middleware, SORA encompasses on-board robot control and a full suite of software tools necessary for remotely operated exploration missions. SORA has been field tested in numerous scenarios of robotic lunar and planetary exploration. The experiments conducted by IRG with SORA exercise a large set of the constraints encountered in space applications: remote robotic assets, flight relevant science instruments, distributed operations, high network latencies and unreliable or intermittent communication links. In this paper, we present the results of these field tests in regard to the developed architecture, and discuss its benefits and limitations.

1 Introduction

Advanced software methodologies are necessary to cope efficiently with the complexity of the software powering any modern robotics system. This need is even amplified for robots

designed for the exploration of uncharted environments since the tasks involved require a high level of autonomy combined with a rich set of interactions with a control team. To address this, the Intelligent Robotics Group (IRG) at the NASA Ames Research Center developed a Service Oriented Robotic Architecture (SORA) to control its exploration robot prototypes. SORA has enabled complex exploration scenarios in realistic environments to be tested while allowing IRG's research in human-robot exploration to smoothly evolve. This paper reports on the lessons learned from more than six years of experiments with the SORA software system.

1.1 Context

Figure 1: K10 Red planetary rover, running SORA during the 2007 Haughton Crater Field Test (Devon Island, Canada). In the foreground, the rover is seen carrying autonomous navigation sensors and science instruments. The base camp, where we simulated an astronaut crew, is visible in the background.

Human-robot exploration of remote locations has been one of IRG's key research topics for more than a decade. This applied research involves field testing to study and validate different system architectures and configurations. Most of these field tests take place in remote locations that serve as Mars or Moon analogs, such as the one shown in Figure 1. Some of the key requirements for the software system running on IRG's robotic platforms are:

- Enable integration of advanced robotic algorithms
- Support a wide range of robots and instruments
- Permit a variety of exploration scenarios to be rapidly tested
- Facilitate interoperability with a whole suite of mission tools
- Allow a small team to perform a variety of experiments supporting a wide range of research priorities in space robotics

To address these challenging requirements, in 2005 IRG began developing the Service Oriented Robotic Architecture (SORA), which is detailed in (Flückiger et al., 2008). SORA is a *software* architecture that is built using proven software engineering development principles and practices (Kroll and Royce, 2005) and leverages the advantages of Service Oriented Architectures. The rest of the paper focuses on this aspect and does not address robot *control* architectures, nor the components necessary to build a robot controller.

Research on software methods and systems for robotics has increased considerably over the past decade. Application of software "best practices" to handle the complexity of robot systems has led naturally to the adoption of software architectures well established in computer science. Three of those architectural paradigms are classified in (Amoretti and Reggiani, 2010): Distributed Object Architecture (DOA), Component Based Architecture (CBA) and Service Oriented Architecture (SOA). This is, however, still a very young domain and despite standardization efforts, such as the Robot Technology Component (RTC) (Object Management Group, 2012c) or the Joint Architecture for Unmanned Systems (JAUS) (Rowe and Wagner, 2008), the landscape of solutions is still extremely fragmented.

Table 1: Acronyms often used in the paper.

SOA	Service Oriented Architecture
DDS	Data Distribution Service
IDL	Interface Definition Language
RMI	Remote Method Invocation
QoS	Quality of Services

1.2 Related work

Robot software frameworks are numerous, each focusing on specific aspects: simplicity, memory footprint, speed, programming language, distribution, robotic domain, etc.

In terms of robotic frameworks, JAUS currently supports the greatest number of robotic systems. JAUS is a messaging architecture aimed at promoting interoperability between unmanned vehicles (ground, air or water). The JAUS standard defines a communication "wire" protocol and sets of messages between nodes, which are organized as a system of four strict functional layers. JAUS provides interfaces to both manipulators and mobile bases for basic control. However, JAUS lacks interfaces that support the rich data communications required by autonomous robots (both for low-level raw sensor data and higher decisional layer level interactions), especially robots used for remote science and exploration.

Some components approaches focus on constructing middleware specifically for robotics. This approach is typified by projects such as OROCOS (Orocos, 2012), or the recent Robot Operating System (ROS) (Willow Garage, 2012). OROCOS provides components targeted to robot manipulators and put the emphasis on real-time controllers. ROS is being adopted at very rapid pace in the robotics research community. ROS provides an excellent collection of robotic algorithms and Operating System type functionality for communication between

distributed nodes. ROS is currently targeted primarily to indoor robots, does not support degraded networks (low bandwidth, loss of signal, etc.), and does not scale well for complex scenarios involving highly distributed communications. The Player project (Collett et al., 2005) is also very popular in the academic community due to its ease of use, the number of common sensors interfaced, and the numerous tools built around it (e.g. Stage and Gazebo simulators). While Player was initially designed for mobile robotics, the YARP project (Fitzpatrick et al., 2008) was started to support research on humanoid robots. In addition to TCP transport, YARP also supports UDP, which is better suited for networks with long delays. Finally, YARP has some level of interoperability with Player and ROS. However, Player and YARP lack features in terms of network scalability and support for the unstructured environments faced by planetary robotics.

Other approaches are focus on building component frameworks on top of existing middleware. This is the case for ORCA (Makarenko et al., 2006) and Microsoft Robotic Studio (MRS) (Jackson, 2007). ORCA project adopted the ICE (ZeroC, 2012) communication framework. MRS relies on Microsoft the .NET framework. Similar to these approaches, SORA relies on existing middleware: CORBA (Object Management Group, 2004) and the Data Distribution Service (DDS) (Object Management Group, 2012a). However, unlike ORCA, SORA's middleware is compliant to *standards*[1], which guarantees longer term support for the project. Despite the fact that MRS relies on the .NET standard, this proprietary framework offers limited support for Unix based systems.

A third approach for robot software frameworks focus on being middleware independent. This is the case for Genom (Mallet et al., 2010), which provides a complete framework for the definition and generation of robotic "module" that constitute the functional layer of a robot controller. Genom has been associated with a component based framework to generate controllers, which are then validated by construction (Bensalem et al., 2010). In addition, Genom has been coupled with planners to obtain robot controllers with a deliberative layer (Ceballos et al., 2011). As with SORA, Genom targets the space robotics domain, but focuses on the robot controller, rather than supporting the full deployment of autonomous systems (rover and ground control).

In terms of NASA frameworks, it is important to mention the Coupled Layer for Robotic Autonomy or CLARAty (Nesnas et al., 2003). CLARAty is a software environment for the infusion of robotic algorithms into future space missions. CLARAty defines a two-layer control architecture, but does not provide a software architecture to assemble the various algorithms. Instead CLARAty promotes the concept of generic interfaces to common robot concepts. The interfaces are directly expressed as C++ classes and dependencies on third party libraries are kept minimal by design. In comparison, SORA does not enforce classes relationship, provides language neutral interfaces that can be mapped to various languages, and relies heavily on middleware.

Finally, the Mars Science Laboratory (MSL) rover and its Mars Exploration Rovers (MER) predecessors are examples of highly reliable flight software (Volpe, 2003; Reeves and Snyder,

[1]Both CORBA and DDS are defined by the Object Management Group.

2005; Biesiadecki and Maimone, 2006). MSL and MER flight software are closed systems, essentially written in C and running space hardened computers[2] powered with a hard real-time Operating System. Current flight guidelines prevent use of modern language features (like inheritance or templates) or higher level Operating System functionalities. The design constraints for these systems are quite different than for SORA in terms of level of effort, target system, communication pattern, etc.

To summarize, SORA has several common characteristics with Component and Service Based Architectures in robotics, but also has several unique features:

- SORA supports a range of robotic scenarios (full autonomy to full teleoperation)
- SORA supports a variety of deployment configuration (local field test to multi-center operations)
- SORA extends well beyond the robot controller and is used across the whole mission tool suite
- SORA has been used extensively in high-fidelity robotic mission simulations

1.3 Experience with Exploration Robots

In (Flückiger et al., 2008) we have described how SORA supported a Lunar analog robotic field test during Summer 2007 at Haughton Crater, Devon Island (Canada). This first full deployment of SORA involved two K10 rovers performing systematic site surveys on a site above the Arctic circle (Fong et al., 2008a). Since then, SORA has been used for analog mission simulations in 2008 at Moses Lake (WA), 2009 at Black Point Lava Flow (AZ) (Deans et al., 2009) and 2010 at Haughton Crater site (Fong et al., 2010), as well as in field tests conducted at sites closer to the NASA Ames Research Center. Table 2 summarizes the mission simulations and instruments that SORA supported during these field experiments. The table highlights the flexibility and scalability required by SORA to address all these scenarios with a small research team.

These field experiments included scenarios covering different possible phases of human-robot exploration in space: site survey and resource prospecting before human arrival, robotic reconnaissance while humans are present and robotic follow-up once humans have departed. Each of these unique opportunities allowed IRG to test SORA in applied scenarios. These scenarios involved full control and science teams, who depended on the robot mobility and data gathering. Field experiments also test SORA service interactions (onboard the robot and to ground control) in situations with non-homogeneous networks including satellite links with variable Quality of Service (QoS) and high latency (0.5s physical to 50s simulated).

[2]Space hardened processors have capabilities approximately equivalent to "earth" processors 10 year older.

Table 2: Summary of key field tests performed with SORA. Most IRG field tests share common requirements in terms of robot platform and science instrument integration, variety of operation modes and interoperability with other tools. In addition, each field test stressed a particular aspect of the software architecture. The SORA capabilities most exercised by each test are highlighted in italics in the third column of the table.

Location and Year	Configuration	Main Objective *SORA key effort*	Primary science instruments
Haughton Crater, Devon Island, Canada, 2007	2 x K10s + small ground team	Systematic site survey (Fong et al., 2008a) *New robot support and new instruments integration*	JPL CRUX Ground Penetrating Radar, Optech ILRIS-3D LIDAR and Microscopic Imager
NASA Ames, CA, 2007	K10 + small science team	Resource prospecting (Fong et al., 2008b) *Rapid prototype with flight instrument*	HYDRA Neutron Spectrometer
Moses Lake Sand Dunes, WA, 2008	K10 + ground team	Systematic site survey and Robotic Reconnaissance (Fong et al., 2008b) *Heteregoneous robot fleet*	GSSI SIR-3000 Ground Penetrating Radar
Black Point Lava Flow, AZ, 2009	K10 + control team	Robotic Reconnaissance (Deans et al., 2009) *New concept of operations*	Optech ILRIS-3D LIDAR and GigaPan camera system
Haughton Crater, Devon Island, Canada, 2010	K10 + control team and science team	Robotic follow up (after human mission) (Fong et al., 2010) *Large scale science experiment*	Mala Ground Penetrating Radar, Optech ILRIS-3D LIDAR, X-Ray Fluorescence Spectrometer and multiple imagers
JSC Rockyard, TX, 2012	Centaur-2 + drivers and ground team	Tele-operated lunar rover navigation using LIDAR (Pedersen et al., 2012) *SORA deployment on a non-IRG robot*	Velodyne HDL-32E
Basalt Hills, CA, 2012	K-Rex + support team only	Mapping for navigation with 2 different approaches (Ames and CMU) *Fast paced data collection field test*	Engineering Field Test: Velodyne HDL-32E and Stereo Cameras

1.4 SORA as Mission Backbone

Figure 2 illustrates a typical field test where SORA is used across the full system deployment: within rovers, by the field team supporting the robots, by the control team executing remote operations and by the science team using analysis and mission planning tools. Whenever SORA services are colocated, or distributed across a network, their interactions are based on unified interfaces and are transparently optimized by the supporting middleware.

In the following, Section 2 describes the high level concepts of SORA. Then, using specific examples the paper highlights the benefits of the SORA in Section 3 and the shortcomings of SORA in Section 4. Finally the paper concludes with future extensions to the current research.

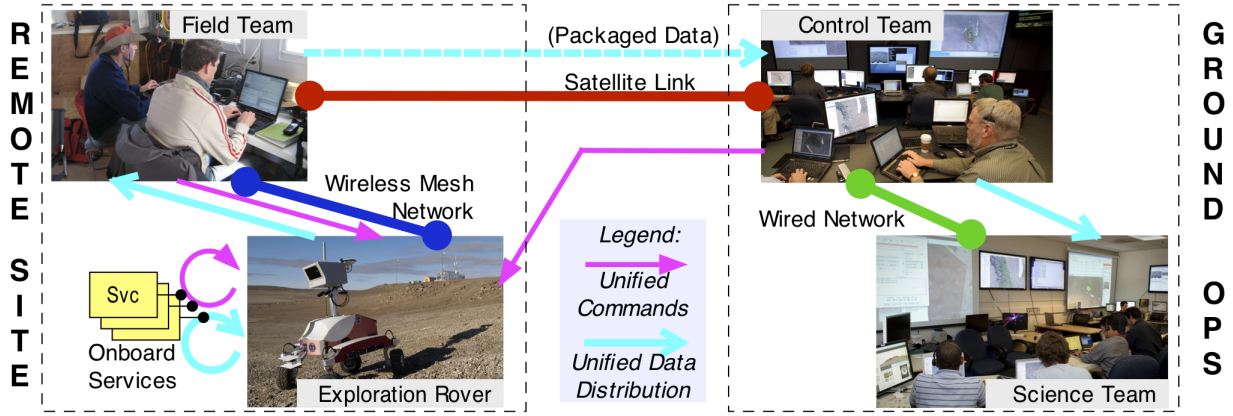


Figure 2: Typical deployment of SORA for a field test involving a Remote Site and Ground Ops. Four nodes are shown: Exploration Rover, Field Team emulating astronauts, ground Control Team and ground Science Team. Common unified interfaces are shared at each node. Data distribution appears identical at all nodes despite the different physical networks. Note that the Science team communicates intents using plans that are transferred to the Control team who upload them to the robot.

2 SORA Concepts

First, it is important to emphasis that SORA is a *software* architecture supporting robotic systems, and does not define a particular robot *control* architecture. The current control architecture of IRG robots is constructed as a two tiered system with hardware and functional components. Our robot controllers are using plan sequencer engines combined with behavioral services. Thus, the control architecture can be considered as a *hybrid* control architecture. However, the SORA approach does not directly reflect the structure of the robot *control* architecture.

SORA applies state-of-the-art software architecture principles and practices to the space robotic domain. SORA embraces the typical concepts of service oriented systems: encapsula-

tion, communication patterns based on stable interfaces, and reliance on robust middleware. This builds a loosely coupled and highly cohesive system.

Figure 3 illustrates a simplified controller constructed with SORA. The details of the services internal structure, and unified services communication modes are described in (Flückiger et al., 2008). The key concepts of SORA are briefly highlighted in the following section, starting with the common characteristics shared by SOAs and finishing with the SORA specific robotic aspects.

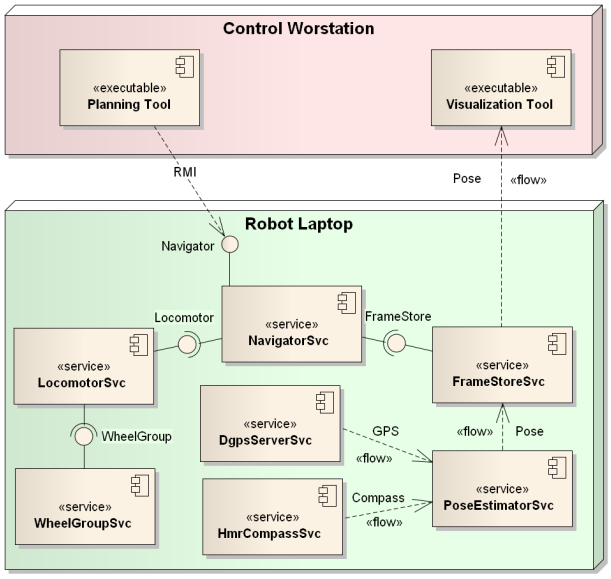


Figure 3: A minimal robot controller, built with only a few services, is represented using the Unified Modeling Language (UML) notation. Data distribution is unified across the full deployment. Provided interfaces are transparently accessible within a robot or remotely.

2.1 Essential SOA aspects of SORA

2.1.1 Services

SORA services encapsulate a set of interconnected classes to offer high level functionalities to the system. Each service is self-contained and is dynamically loadable. In addition, a service manages its own control-flow requirements (message loops, threads etc). A service can be passive, just waiting for events, or active with one or multiple threads of execution.

2.1.2 Interfaces

Strongly typed, network transparent interfaces, specified with the Interface Definition Language (IDL) (Object Management Group, 2012b), allow connecting to the services. Implementation of the interfaces in different languages allows heterogeneous systems to interact.

The same control interfaces are accessed using Remote Method Invocation (RMI) for interactions between services on the robot as well as by applications running off-board.

2.1.3 Data Distribution

In addition to RMI, SORA uses a publish/subscribe scheme to distribute data across services, within a single controller, and to the ground control systems. SORA initially used the CORBA Notification Service (Object Management Group, 2004) to implement a publish/subscribe mechanism. This implementation has been since replaced by the DDS for better scalability. DDS is an Object Management Group standard based on publish/subscribe data messaging. All DDS messages are also described with IDL description files. Source code to manage the corresponding data structures is then automatically generated for the target language (C++ and Java in the case of SORA).

2.1.4 Middleware

SORA relies heavily on middleware, specifically the ACE/TAO implementation (ACE/TAO, 2012) of the CORBA standard. In addition to CORBA, SORA uses the MIddleware for RObots (Miro) (Utz et al., 2002). Miro facilitates the use of CORBA in the robotics context, without introducing an extra layer of indirection, but by providing a configuration of the middleware tailored to the robotics domain. In addition to CORBA, which is used for commanding, SORA relies on the RTI (RTI, 2012) DDS implementation for data distribution. DDS provides several key capabilities that CORBA does not have, including QoS management for data distribution across heterogeneous, unreliable networks.

2.2 Robotic domain aspects of SORA

2.2.1 Services Assembly

A robot controller is constructed from a set of services. These services are started according to a configuration file crafted for a particular operational scenario. The same configuration mechanism is used also for services not running on the robot, such as simulated components.

SORA uses the "Component Configurator" pattern (Schmidt et al., 2000) to combine the services in a full system. A configuration file specifies which services should be started to create a particular controller.

2.2.2 Standard messaging between NASA robots: RAPID

A key design goal of SORA was to generalize interfaces and data-structures for use on a broad set of robotic platforms. In addition, from its inception, SORA was designed to use a publish/subscribe model for the distribution of robot data, within the robot controller itself, and to external consumers as well. Based on experience gained using SORA over

several years, a collaboration with other NASA centers led to the development of standard messages for NASA robots. The resulting RAPID project (Torres et al., 2009) created messages derived from the CORBA based messages of SORA, but with modifications and extensions to satisfy the constraints for the robots from three NASA centers: NASA Ames (K10s and K-REX rovers), Johnson Space Center (LER, Centaur-2 rovers) and the Jet Propulsion Laboratory (Athlete 6-legged robot). RAPID consists of a set of IDL files plus utilities classes. RAPID uses the DDS as middleware and makes ample use of the advanced feature set of the data-centric publish/subscribe model of DDS.

RAPID is built upon a lean set of design concepts that are consistently applied throughout the message set. A core design goal was to provide a rich data-representation applicable to a wide variety of robotic assets, while keeping bandwidth usage low. This is achieved through the separation of data-streams into three categories of messages: *Configuration*, *State*, and *Sample* messages. Typically state or samples messages types are paired with a configuration message type. The configuration messages carry information that helps the interpretation of the state or sample message, and are sent only once unless some configuration parameter changes.

RAPID Configuration messages. Configuration messages are used to describe the characteristics of robotic subsystems. For instance the properties of laser range scanners, such as the number of scan-lines, width of a scan-line, or the distance resolution differ significantly by model. This information is published as a *configuration* message. This allows software modules to publish the laser range data in a generalized, concise data-format with *sample* type messages (see below). The configurations are published only once as `reliable` and `durable` messages. These two QoS associated with the configuration messages mean respectively: every subscriber is guaranteed delivery and a subscriber joining after the single configuration message was sent will still be notified of this message. Due to their low publication frequency, configuration data types are allowed to be verbose and thus can have a large footprint.

RAPID State messages. Many robotic subsystems switch between discrete states (active vs inactive, operational vs failure, etc.) at irregular intervals. It is critical that these state changes are always observable by other subsystems or operators. Therefore, messages representing subsystems states are modeled as *state* messages, which are `reliable` and `durable` (same QoS than configuration messages). As state-changes can come in higher frequency or in bursts, their footprint is kept minimal (unlike configuration messages). Furthermore, the middleware is only required to keep a very limited history of those publications. This enables skipping the delivery of some state updates to subscribers in the case of network congestion or Loss of Signal (LOS). These situations are common for space communications links.

RAPID Sample messages. Sensor data published at a fixed, high frequency produces most of the data on exploration robots. This data encompasses raw sensor data, such as pose sensors, image sensors, laser scanners, as well as derived data, such as pose estimates,

point-clouds or maps. Typically, such publications concern continuously changing properties (location, proximity of obstacle, etc.) where the latest data invalidates the previous reading. This category of data is modeled in RAPID as *sample* messages. Due to their potentially high rate, these messages are designed to minimize their footprint. Also, their publications are typically designed as `best-effort` delivery. This QoS means that messages are not guarantied to be delivered and that there is no re-sending of previous publication to late joining subscribers.

2.2.3 Supports communication across the mission

SORA application domain extends well beyond the scope of the robot controller. The unified interfaces and communication patterns are designed to support mission scenarios from ground control tools to rover internals as well as development and debugging tools.

Figure 4 shows two software tools that are extensively used during our field test: xGDS and VERVE (Lee et al., 2012). xGDS is a Ground Data System to support scientific missions. It allows scientists to collect, archive and analyze science data, as well as to create high level plans. VERVE is a powerful interactive visualization tool providing real-time visualization of most of the data the rovers are generating. The latest incarnation of VERVE also allows sending critical commands to the robotic subsystems. Both tools interoperate with the rover software on various robots using SORA.

Figure 4: SORA provides a common communication system for multiple robotic platform and mission tools. The upper image shows three rovers supported by SORA: K10s, K-REX and Centaur-2. The lower left image is a screenshot of the xGDS (Ground Data System), and the lower right shows a screenshot of VERVE (interactive 3D visualization tool).

2.2.4 Validation with field tests

As shown in Table 2, SORA has been deployed in multiple field tests on various rovers. In addition to the K10 series rovers, SORA has been tested on a smaller scale rover, K10-mini (footprint of 40cmx30cm), as well as the much larger new IRG rover K-REX (footprint 2mx1.6m). Outside of IRG rovers, SORA also currently supports a navigation system based on LIDAR (Pedersen et al., 2012). The range of situations encountered across these field tests is summarized in Table 3.

The unified interfaces across the system, for both colocated and distributed scenarios, facilitates re-use and loose coupling. Similarly, the facility to easily create specific robot controllers for particular scenarios by assembling different sets of services, permits the agile deployment of SORA to field tests. Scalability is obtained by the service encapsulation, the loose coupling between services, and the interchangeability of services providing identical interfaces. Finally, insulation of each service and reliance on robust middleware promotes a high level of reliability. These quality attributes for SORA are further detailed in the next section, which also discusses the performance of the architecture.

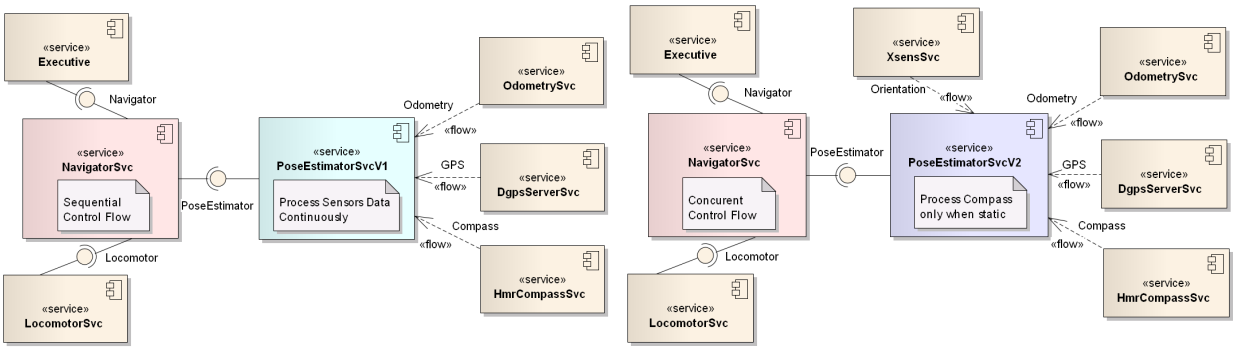
Table 3: Range of key parameters during field tests using SORA

Parameter	Minimal/Unfavorable configuration	Full-blown/Optimal configuration
Configuration	1 robot + field team of 5	2 robots + field team of 6 + ground team of 9 + science team of 12
Local wireless network	10Mbps (degraded 802.11b) to no comms (robot out of range for extended periods and navigating fully autonomously)	50Mbps (Meshed Tropos network with 802.11n)
Link to ground	1-2Mbps (artificially constrained or satellite link) to no comms (link loss or no ground team)	15Mbps (microwave link) with optional 50s delay introduced
Number of services on the robot	Simple navigation: 12 [Hardware (HW)=2, Software (SW)=6, IN (Infrastructure)=4]	Autonomous navigation and science instruments: 55 [HW=19, SW=22, IN=14]
Distributed services (not on the robot)	1-2 ("mission manager" to start an autonomous plan and monitor robot health)	$>$ 5 multiple control panels and 3D visualization plus data collection system
Data collected on the robot	80MB/h (no science and exclude stereo images)	1GB/h (LIDAR data + stereo images included)

3 SORA Benefits

This section describes the benefits that SORA brings to the IRG robotics field tests. Several advantages reported below are derived from SOA specific concepts like encapsulation, communication pattern, and exposition of stable interfaces. In addition, the use of a com-

ponent configurator pattern and reliance on robust middleware increases the flexibility and reliability of the system.



(a) HMP 2007 Controller

(b) HMP 2010 Controller

Figure 5: Two successive versions of the K10 controller represented with UML (only few services shown). The new `PoseEstimatorSvc2` from Figure 5b subscribes to an additional sensor stream (`XsensSvc`) compared to the `PoseEstimatorSvc1` from Figure 5a. However the Pose Estimator continue to expose the same interface, thus requiring no changes for the `NavigatorSvc`.

3.1 Flexibility

3.1.1 Stable Interfaces

All SORA interfaces (allowing remote method invocation), and all data structures (participating in the publish/subscribe mechanism) are defined using IDL. Each IDL specification is designed to be as generic as possible while allowing access to specific capabilities of the subsystems. The current interfaces are built upon the expertise IRG gained with previous systems: CLARAty C++ interfaces, the Mission Simulation Facility HLA based interfaces (Flückiger et al., 2005a) and the ICE based interfaces developed for the Peer-to-Peer Human-Robot Interaction experiment (Fong et al., 2005). In addition, the RAPID specification is a collaborative work between ARC, JSC and JPL, with the goal of offering a standardization for common robot messages. These interfaces and data structures have certainly evolved from the initial SORA conception to today's system. However, changes are mostly extensions of existing interfaces or addition of new data structures to address a new domain. For example RAPID defines a core message set (shared by NASA robots) to which specific applications can add their own extensions. Keeping these interfaces stable and their specification in a unique repository (shared by all the parties contributing to software for robotic field tests) makes it possible to easily swap a service for an equivalent one and makes it easier to maintain the tools around the robot controller.

An example of this evolution is shown in Figure 5. A new version of the `PoseEstimator` was written for the 2010 HMP field test. The `PoseEstimator` computes the best estimate of the rover position and orientation using a Kalman Filter to process various sensor inputs (not

all included in this figure). The second version of the `PoseEstimator` relies on an additional sensor, and computes poses using a new algorithm. Thanks to the SORA architecture, the previously existing sensor data is consumed the same way and the services dependent on the PoseEstimator did not have to be modified at all.

3.1.2 Component Configurator Pattern

The component "Component Configurator" pattern (Schmidt et al., 2000) enables SORA to change scenarios without changing any code. The same code base is used and re-used for different scenarios simply by creating different configuration files.

Despite the fact that these controller configuration files are currently crafted manually, they have been a tremendous tool to develop and test our robotic software. Configurations are created for each individual scenario. Scenarios range from a minimal controller (containing only the locomotion service) to a full blown field test controller (requiring a suite of science instruments). In addition to facilitating development, the simplicity and rapidity of creating a new controller configuration also allows optimizing controllers based on resource usage or memory footprint. The flexibility and robustness of the SORA services assembled with the component configurator pattern can be measured by the number of services (30 to 50) running in concert on a robot, while sharing the resources harmoniously.

A robot controller assembled for a typical exploration scenario with autonomous navigation and a few science instruments averages 45 services. These services can be grouped into three categories:

1. Hardware: an average of 14 hardware services are in charge of communication with physical sensors and actuators.
2. Software: an average of 19 software services are in charge of data processing and high level algorithms for autonomy and control.
3. Infrastructure: an average of 12 services are performing infrastructure tasks ranging from audible notifications to bandwidth management.

Combining these services to obtain a controller adapted to a particular scenario can be done in minutes, which brings a great agility to our robotic platform.

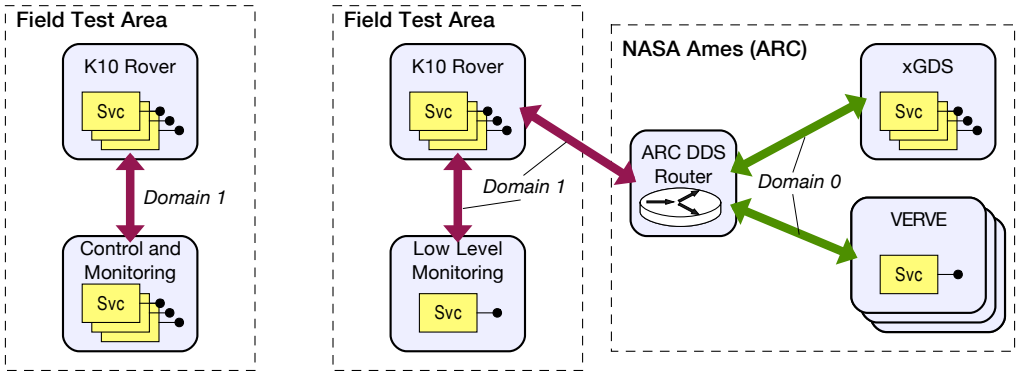
3.1.3 Service configuration

To address the configuration of individual services (not the assembly of services), SORA relies on the Parameter Framework provided by Miro. This overcomes a limitation of the Component Configurator pattern implemented by ACE, which only provides command line equivalent for configuration. The Miro Parameter Framework allows a bi-directional mapping of parameters between an XML file and a service. For example, the *Locomotor* parameters

(mechanism dimension, speed and acceleration values, etc.) for a specific robot is loaded from a description file and applied to the service with minimal coding. The Miro Parameter Framework generates code and GUI editors for a particular set of parameter from a Schema definition. The tool saves development time and reduces error sources in the configuration process, such as syntax errors or typos in configuration files.

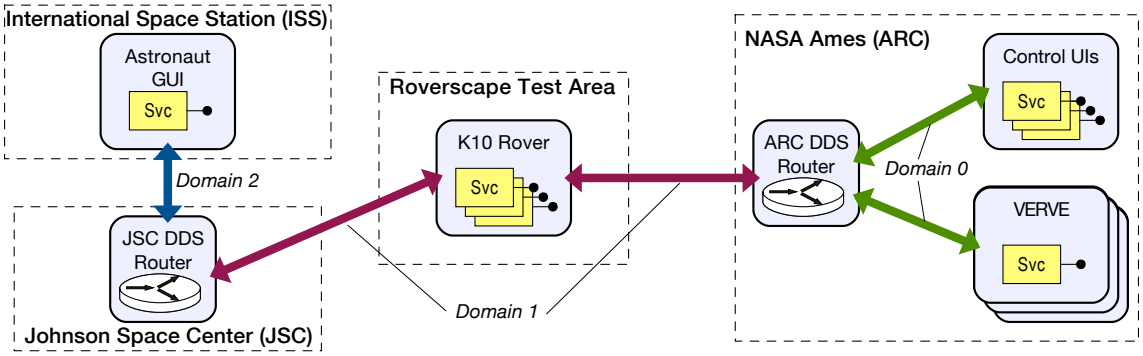
3.1.4 Topology of services

SORA's distributed architecture is designed to support a range of scenarios that require different network configurations. This is illustrated in Figure 6 for three different scenarios. For example, SORA can support a single robot controller for an autonomous system with minimal ground control interactions, as well as a fully distributed scenario with many subscribers of large amounts of data. When tighter control of network traffic is required, DDS allows partitioning of publishers and subscribers into different domains, and provides routing to bridge domains. The publishers and subscribers on a particular domain are completely isolated from publishers and subscribers on another domain. A router subscribes to a set of messages and re-distributes them for subscribers on a separate domain. The router can be configured to select which messages should be re-distributed and at which frequency.



(a) Simple deployment with local control only

(b) Deployed remote rover and ground control system on separate sites



(c) Surface Telerobotics configuration spanning over 3 DDS Domains

Figure 6: Variations of SORA network topologies.

3.2 Scalability

3.2.1 Abstract Interfaces

Abstract service interfaces help considerably to reduce link time dependencies to other subsystems and libraries. For example, when we previously developed an application using CLARAty, the C++ high-level interfaces drew in more than 40 conceptually unnecessary library dependencies [3]. The same application based on SORA's abstract interfaces currently relies on 6 libraries. These libraries contain code actively used by the client implementation.

Abstract interfaces allow services to be easily replaced with different implementations. This enables replacing some (or all) services interfacing with physical components of the robot by simulated components. This facilitates development and testing by not requiring an on-line physical robot. The `Locomotor` service in Figure 3, for example, is responsible for translating high level locomotion commands (translate, drive arc) to individual motor commands. These low level commands are passed to the `WheelGroup` service that abstracts the actual robot hardware. However, a simulated `WheelGroupSim` service, which simulates the robot wheels motion, can simply be started in place of the original service to obtain a simulated rover motion. Applications, such as the 3D visualization tool VERVE (Lee et al., 2012) which is used to monitor the rover progress, do not need to be modified at all. Thus, by simply selecting a different configuration file, we can transparently start a real rover controller or a simulated rover controller.

Just as class polymorphism is a powerful concept with object-oriented languages, interface polymorphism is equally beneficial. SORA uses extensive interface inheritance (supported by CORBA) to abstract services and to increase the scalability of the system. The best example in SORA are the interfaces to science instruments services. For each field test IRG rovers have been equipped with different sets of science instruments to achieve particular science goals. Each instrument has a particular set of characteristics that require specific methods. However, in SORA, all instruments interfaces inherit from the same base `Instrument` interface. This allows a range of services to control and access any instrument in a transparent manner at a high level. This is highly useful for the `Executive` service, which executes plans defined by the scientist. A plan contains instructions specifying when instruments need to be activated/deactivated, or when to acquire a sample. The `Executive` is only aware of the base type of `Instrument` and thus is able to command any instrument that inherits and implements that base-interface.

3.2.2 Communication Patterns

SORA services are interconnected with a dual communication pattern: RMI and Data publish/subscribe. These two modes are complementary. RMI is especially convenient for commanding individual services and querying their state. The publish/subscribe mechanism is

[3] While it is possible to achieve a high-degree of decoupling with pure C++ interfaces, keeping the interface definitions independent enforce the practice of separating interfaces and implementations

better suited for distributing data to multiple services.

Remote Method Invocation. Even though it is possible to implement a request/reply pattern using a data distribution model, the stricter RMI approach enables greater static checking and code generation. Thus RMI makes the implementation of transaction-oriented interfaces, such as robot commanding, more efficient and less error prone. In addition to the regular RMI concept, SORA uses the Asynchronous Method Invocation (AMI) (Schmidt and Vinoski, 1999) pattern which augments services decoupling and simplifies services implementation. For example, it can be impractical for the caller of a service to block its thread of execution while waiting for completion of an operation. Using AMI, the call will immediately return and the caller will be notified by a callback when the operation has completed. In SORA, the complexity of AMI, thread safety, and exception handling is handled by middleware.

The ability to remotely inspect the running system is a side benefit of network transparent high-level interfaces. This remote inspectability is important for scripting, unit testing, and online-supervision of the system in operation. In case of a failure, individual components can be analyzed as part of the running system, significantly reducing the time to locate the issue. This interaction can even be used to work around some of the problems that autonomous system encounter, by allowing human intervention.

Publish/Subscribe. Despite the advantages of RMI and AMI, data distribution is preferable when multiple consumers are interested in the same type of data, or when data needs to be transmitted periodically. A publish/subscribe mechanism decouples services by allowing producers of data to be independent of the consumers. In addition, SORA relies on publish/subscribe to enable data logging. The logger application is a generic data consumer, which can subscribe to any message type and serialize it to file, including a trace of the request/reply pairs of commands.

Currently SORA relies on the *rti-recorder* from RTI to capture all the DDS traffic[4]. Events recorded can then be replayed at a later time with the exact same messages (timing included). Two major capabilities are enabled by this functionality: 1) analyze a particular situation and 2) test a different version of an algorithm with previously collected data. For example, the mapping system used in SORA was primarily developed offline and tested with pre-recorded datasets.

3.2.3 Encapsulation

Encapsulation of robot capabilities into services with well defined interfaces effectively shields the overall system from code modifications within services. As long as the IDLs (for the service interfaces and the data distribution messages) remain the same, any change to the internals of a service will not affect other services. This is illustrated in Figure 5 with the

[4]Previously SORA used the *LogPlayer* tool from Miro when the data distribution was implemented with the CORBA Notification Service

evolution of the `Navigator` service. The navigator service allows the rover to reach a given goal while avoiding obstacles by building a dynamic map of the environment. We can see from the figure that the `Navigator` service has strong dependencies on the `Locomotor` service and `PoseEstimator` service. In addition, the `Navigator` service is used by the `Executive` service. In our work, the navigator has undergone major re-structuring over the years to improve performance and flexibility. In particular the initial navigator relied heavily on navigation classes from CLARAty (Nesnas et al., 2003). The current navigator, however, replaces the previously sequential model with a newly developed, concurrent framework. In the framework, sensor reading, map building and action selection are asynchronous and the robot drives continuously. New terrain analysis and path evaluation algorithms were incorporated into the navigation system. This extensive development effort has been transparent to the many users of the Navigator service.

3.3 Reliability

3.3.1 Middleware

The architectural paradigms implemented in SORA would not have achieved such a reliable and extensive set of features without highly capable middleware. As mentioned in Section 2, SORA heavily relies on CORBA coupled with Miro, and on DDS for data distribution. These dependencies impose some constraints due to the choice of a specific middleware (Section 4). In particular, middleware is pervasive, so replacing any middleware for another one requires substantial code changes. However, this commitment to a set of well-established libraries enables rapid progress with a finite (and usually limited) amount of resources for research and development. During the course of that work, ACE/TAO CORBA has gone through several release cycles. SORA directly benefits from these new versions, which represent a considerable amount of work from outside parties. At the same time, because CORBA is a standard, new revisions of the implementation have only required minor changes to SORA's code. Similarly, DDS implementation improvements free SORA developers from maintaining this large middleware themselves.

In addition, appropriate use of middleware isolates robot software developers from changes in the lower layers. For example, as shown in Figure 2, the Remote Site and Ground Ops are connected using an unreliable satellite link. To cope with lower and intermittent data rates, robot telemetry was transferred using a specific method developed as part of Miro. However, this extension of the data distribution method is completely transparent to the services running either on the Remote Site or Ground Ops. The exact same applications can run on each site, without any need to know what physical link connects the sites.

Finally, existing stable middleware considerably increases the reliability of the system. The CORBA and DDS libraries used by SORA are also used by numerous other projects, including many non-robotic applications. Thus these libraries are exposed to a large user base, which continuously tests the software and identifies defects. This level of testing cannot be achieved with a custom middleware layer developed by a single group. Relying on quality

middleware eliminates most of the potential defects [5] linked to the low level layers and allows robotic researcher to focus on their algorithms.

3.3.2 Code generation and code re-use

From the interfaces and messages definition specified in the IDL, source code to manipulate the corresponding concepts is automatically generated. The code can be generated in multiple target languages and contains all the data structures, client/server stubs and other support classes. Relying on code generators not only simplifies the use of interfaces and messages, but also helps minimize the risk of defects. Rather than validating dozen of interfaces and associated helper code, only the code generator itself needs to be validated.

Organizing the robot control software by services decouples the interactions between various parts of the software, and thus simplifies re-use of existing code. Several services that we developed for SORA have been re-used for most of the field experiments. This is especially true for each hardware abstraction (Compass, GPS, IMU, Wheels) because the same sensors can be used for different scenarios, and this is also the case of higher level algorithms (e.g. the generic Locomotor or global Path Planning). In short, we have found that a SOA, which allows service assembly by configuration file, drastically reduces the time needed to create a new robot controller.

3.3.3 Deployment in multiple contexts

To date, SORA has already inter-connected 6 complex software systems: (1) the robot controller for the IRG rovers, (2) the IRG 3D visualization tool VERVE, (3) the IRG ground data system xGDS, (4) the Johnson Space Center (JSC) user interface PIGI (Burridge and Hambuchen, 2009), (5) the Smart-Spheres on the International Space Station (NASA, 2012), and (6) the Resolve payload by Kennedy Space Center (KSC) (TwinOaks Computing, 2011a). To support these various software systems, two different implementations of CORBA have been used: ACE/TAO and JacORB (Brose, 1997). Similarly, two different implementations of DDS have been employed: RTI and CoreDX (TwinOaks Computing, 2011b). These applications demonstrated good middleware vendor inter-operability. SORA infusion in such complex systems has been greatly facilitated by the reliability of the underlying middleware.

3.4 Performance

SORA is a software architecture. Therefore its performance is measured on how well it supports robot software development, not how well the robot control system performs.

In terms of data passing within the robot and outside, our most recent field test performed with the K-REX rover in 2012 at Basalt Hills provides some meaningful numbers. During

[5] Few defects in the middleware are still un-earthed by SORA because our scenarios are pushing the communication boundaries beyond mainstream applications. This is typically due to the specificities of the space robotic domain, like extreme time delays.

this test, K-REX performed 5 traverses totaling about 11km of driving on rough terrain on multiple levels of a rock quarry. The rover controller collected data from the full vehicle odometry at 25Hz, two Inertial Navigation Systems (INS) (20Hz), stereo cameras at full resolution (two times 1.4 Megapixels at 2 Hz), and a 32 beam laser range finder (about 700,000 points per second). In addition, the rover controller created mapping products in real-time. In addition to data transfer within the controller, most of the data was pushed to a remote control station. Timing of the various components on the rover revealed that the overhead of message passing is minimal compared to autonomy algorithms (mapping and localization).

Static type definition of data structures allows for precompiled serialization, which can be optimized for given criteria. In addition, DDS and CORBA allow for shared memory data transport, which allows optimized message passing when the services are collocated. Although shared memory transport introduces a small overhead compared to directly passing messages with pointers, we consider this cost negligible compared to the benefits of relying on a safe, stable, and scalable middleware.

Finally SORA exploits QoS management provided by DDS. With DDS, multiple QoS can be combined in the same system by appropriate configuration. This is leveraged by the different type of RAPID messages (Section 2.2.2), allowing an optimal allocation of the bandwidth. To further improve the tuning of the bandwidth, DDS offers routing services. Routing allows redistribution of data of selected topics at specific frame rates on different domains. This capability isolates network traffic and allows control of bandwidth usage on the various parts of the network. For example, Figure 6c illustrates the network configuration for the 2013 Surface Telerobotics experiment (Bualat et al., 2012) where astronauts on the International Space Station will control a K10 rover at Ames. In this complex scenario involving two NASA centers and a space asset, three DDS domains are configured with routers tuned to guarantee that bandwidth allocations will be honored.

More advanced DDS QoS features have also been leveraged for NASA field tests. For example, we demonstrated the full operational capabilities of a robotic scouting task in presence of lengthy (50 second) delay between the robot and the ground control. Again, achieving this level of performance would not have been possible without highly capable middleware.

3.5 Software Agility

Starting in 2007, the IRG conducted three significant field experiments with the K10 rovers over an 18 month period. Each experiment had a very different mission scenario with different instrument payloads. In addition, a major hardware revision, including motors, motor controllers and kinematic parameters, was performed before each of the experiments. Finally, significant capability improvements were incorporated, including a continuous navigation system, which replaced the previous "stop-and-go" navigation. Our architecture allowed all of these transitions to be handled without rupture in our rover usage.

We have found that the adoption of the SORA architecture has greatly boosted developer

productivity. In the 6 months preceding the Haughton Crater 2007 field test, the four person team working on the rover software was able to drastically improve the locomotion and navigation system, integrate new science instruments (GPR, LIDAR), develop ground control tools, and implement a complete mission scenario. The service architecture not only benefited the core rover team, but all parties providing software for the same field test, by providing them with a robust platform for software integration. Since then, one or two field tests have taken place annually, supported only by a 2 person rover software team.

The latest field test that we performed at Basalt Hills (CA) in late 2012 demonstrated additional advantages of using RAPID and SORA. This field test was conducted in collaboration with the CMU Robotics Institute who demonstrated their mesh Reliable Autonomous Surface Mobility (RASM) (Wettergreen and Wagner, 2012) mapping system on the K-REX rover. For this test, the inputs and outputs of the RASM software were adapted to use the RAPID messaging system. The adaptation was performed by a student, who had no prior exposure to SORA and DDS, in less than 6 weeks. This period included the creation of RAPID interfaces to the RASM system, tuning of the algorithms for the K-REX robot, and one week of field testing.

4 SORA Shortcomings

The long-term use, continuous development, and intensive field testing of SORA has provided us with significant insight into SORA's design, implementation, and software technologies. In this section we discuss some of the limitations that we have identified: re-use of data structures, synchronization of services, and middleware acceptance by external parties.

4.1 Rigidity of Data Structures

The publish/subscribe model of data-distribution is central to much of SORA. Unfortunately this model violates some of the abstraction concepts of object-oriented design. The data-structures used for distributing information through the system become the public interface to write applications against. This is a necessary caveat in a data-centric distributed applications such as robotics, but can affect maintainability and code re-use.

In addition, the data-distribution systems used by SORA do not efficiently support type-polymorphism. A data-bus supporting single-inheritance in the disseminated data-structures would allow generic data-consumers to subscribe to a generalized concept (e.g. position), while ignoring the sensor specific information (e.g. additional GPS data fields). However, the CORBA Notification service, as well as DDS, only allow retrieval of the message content as generated by a message publisher.

In consequence, SORA data producers, such as a pose sensor, are less interchangeable than if type-polymorphism was available. In a similar way, code re-use is limited when writing data consumers since they cannot share a common high-level data type. Finally, this limitation

also affects the maintainability of data collected during field tests. The logged data is used extensively after the field tests for analysis and development purpose. So any extension of previously defined data-types requires additional effort to convert the logged data to match the new definitions.

4.2 Middleware Acceptance

The acceptance of specific middleware, especially of CORBA is often an issue. The major issues reported usually are footprint and complexity.

Both those arguments are only partially true. Middleware packages usually have similar footprint as other frameworks and libraries that are regularly used in the development of large-scale systems (GUI toolkits, data-bases, JIT-compiler etc). However, it is difficult to over-come established misconceptions regarding the complexity and performance of certain middleware (e.g. CORBA). In our experience, middleware can be managed by a small number of domain experts, without requiring all developers to cope with complexity.

One key factor is that most middleware packages (especially open-source packages like ACE/TAO) are not trivial to install and to integrate into the build-process. ACE/TAO now provides packages for most Linux distributions, but an installer for Windows is still missing. Also, auto-generated code can have poor readability, which makes it difficult for the non-domain expert to understand.

4.3 Synchronization of Services

In a loosely coupled architecture, tight synchronization of services is generally not envisioned. This is generally true for most of our services. Triggering a service activity on an event emitted by another service is straightforward, but other synchronization primitives do not exist in SORA.

This becomes more of an issue with simulation, when single-stepping and faster-than-real-time execution are needed. Synchronization of systems timing is usually provided by network services. But a uniform, synchronized time-step is difficult to provide efficiently in a large-scale distributed system and generally not provided by any middleware, or object model infrastructure. Currently, service synchronization is not a SORA requirement. But if more time critical simulators are introduced in the SORA system, we might consider developing synchronization mechanisms similar to those provided by the High Level Architecture, which we have used for previous projects (Flückiger et al., 2005b).

5 Future Work

Future work on the SORA core software includes refining some of the SOA concepts, continuing to standardize the RAPID robotic interfaces with other NASA centers and integrating

new algorithms. In addition, to infuse SORA technologies and concepts into space missions, we would like to quantify the benefits of SORA for its application domain. This would require the identification and application of metrics to measure the system. Some metrics could include the man power required to maintain and evolve the system, the percentage of the code base that is changed for each new mission, or the meantime between failures (recoverable or terminal) for each experiment. Other metrics could include defect rates or complexity analysis. In particular, we plan to perform static analysis of SORA by collaborating with another group at NASA Ames (Venet and Lowry, 2010). Exposing the system to a broader community would also help gather quantitative and qualitative measurements. In this spirit, SORA source code has been already cleared for release under the NASA Open Source Agreement licensing, and thus is available to a larger community for evaluation and contributions.

As we have discussed, SORA has been extensively tested during terrestrial field experiments, and we would like to find a path to inject this software system into flight missions. A step in this direction has been already taken with two new projects (Fong et al., 2012) involving operations with the International Space Station (ISS). Table 4 summarizes the goals of the Smart SPHERES and Surface Telerobotics projects. Both of these projects are using SORA concepts and RAPID messaging as the communication backbone. Utilizing our software for operation with the ISS introduces a number of requirements in terms of operational constraints, astronaut training, software certification and communication with space assets. The deployment of SORA and RAPID on ISS will help further mature the software architecture.

Table 4: Ongoing projects supported by SORA and RAPID.

Smart SPHERES	Surface Telerobotics
The volleyball sized free-flyers SPHERES have been aboard the ISS since 2006. They now are equipped with Android based smart phones running a high level controller. Astronauts aboard ISS will define tasks for the Smart SPHERES using a VERVE based interface. Tasks can include environmental survey, inventory checking or imagery collecting. In return, SPHERES will produce RAPID telemetry and sensor data available both to the astronauts and on the ground.	A K10 rover located at NASA Ames will be tele-operated from the ISS to simulate a Lunar surface robotic operation from L2 Earth-Moon Lagrange point. This type of activity would reduce communication requirements and maximize robot utilization, thus lower mission cost. The mission is to deploy a film based radio-telescope antenna on the ground. The astronaut will command and supervise high level tasks performed by K10 running SORA. Telemetry between ground and ISS will be using RAPID.

6 Conclusion

In this paper, we have described the design concepts underlying the Service Oriented Robotic Architecture (SORA), the benefits brought by this approach and the difficulties encountered.

SORA has been deployed to multiple high-fidelity mission simulations of remote rovers controlled from ground operations. These experiments have demonstrated the advantages of SORA in terms of flexibility, scalability and reliability. At the same time, these experiments have helped identify the limitations of our approach.

The lessons learned during the past 6 years of evolving SORA can be summarized in three points. First, the existence of SORA is the recognition that the software performance of a robotic system is not only due to its control architecture, but that the software architecture also plays a key role. From its inception, SORA has been built with sound software practices, using a modern software architecture and leveraging proven middleware. Second, the standardization of messaging for robots, taking into account the space domain constraints, is key to the usability and sustainability of the robotic assets. RAPID, a lean, yet powerful messaging system opens collaborations between research groups, and facilitates infusion into flight systems. Finally, the advantages of SORA extend beyond the domain of robot control. SORA serves as the backbone supporting IRG's field testing by connecting various robotic mission tools with a powerful distributed system infrastructure. The SORA design and implementation has enabled a full ecosystem of robotic capabilities, and will continue to support their further evolution in the future.

Acknowledgments

This work was supported by the NASA Exploration Technology Development Program, the NASA Enabling Technology Development and Demonstration Program, and the NASA Game Changing Development Program. The authors would like to thanks all the individuals who contributed to the SORA system: Mark Allan, Xavier Bouyssounouse, Matthew Deans, Laurence Edwards, Susan Lee, Mike Lundy, Eric Park, Liam Pedersen, and Vinh To, as well as the numerous interns who spent time on this project. In particular, we are thankful to Terry Fong, IRG lead, who has always been so supportive of this effort.

References

ACE/TAO (2012). ACE, TAO, and CIAO sucess stories. `http://www.cs.wustl.edu/~schmidt/TAO-users.html`.

Amoretti, M. and Reggiani, M. (2010). Architectural paradigms for robotics applications. *Advanced Engineering Informatics*, 24(1):4 – 13. Informatics for cognitive robots.

Bensalem, S., de Silva, L., Gallien, M., Ingrand, F., and Yan, R. (2010). "rock solid" software: a verifiable and correct-by-construction controller for rover and spacecraft functional levels. In *International Symposium on Artificial Intelligence, Robotics and Automation for Space, Sapporo, Japan*.

Biesiadecki, J. J. and Maimone, M. W. (2006). The mars exploration rover surface mobility flight software driving ambition. In *Aerospace Conference, 2006 IEEE*.

Brose, G. (1997). JacORB: Implementation and design of a Java ORB. In *DAIS'97, IFIP WG, International Working Conference on Distributed Applications and Interoperable Systems*.

Bualat, M., Deans, M., Fong, T., Provencher, C., Schreckenghost, D., and Smith, E. (2012). ISS crew control of surface telerobots. In *Proceedings of IAF/AIAA Global Space Exploration Conference*.

Burridge, R. R. and Hambuchen, K. A. (2009). Using prediction to enhance remote robot supervision across time delay. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5628–5634. IEEE.

Ceballos, A., Bensalem, S., Cesta, A., De Silva, L., Fratini, S., Ingrand, F., Ocon, J., Orlandini, A., Py, F., Rajan, K., et al. (2011). A goal-oriented autonomous controller for space exploration. *ASTRA*.

Collett, T. H., MacDonald, B. A., and Gerkey, B. P. (2005). Player 2.0: Toward a practical robot programming framework. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2005)*.

Deans, M. C., Fong, T., Allan, M., Bouyssounouse, X., Bualat, M., Flückiger, L., Kobayashi, L., Lee, S., Lees, D., Park, E., Pacis, E., Pedersen, L., Schreckenghost, D., Smith, T., To, V., and Utz, H. (2009). Robotic scouting for human exploration. In *AIAA Space 2009*, Pasadena, California.

Fitzpatrick, P., Metta, G., and Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous systems*, 56(1):29–45.

Flückiger, L., Neukom, C., Pisanich, G., Buchanan, E., Wagner, M., and Plice, L. (2005a). Experiments with autonomous software for planetary robots: A simulation success story. In *'i-SAIRAS 2005'-The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, volume 603, page 68.

Flückiger, L., Neukom, C., Pisanich, G., Buchanan, E., Wagner, M., and Plice, L. (2005b). Experiments with autonomous software for planetary robots: A simulation success story. In *i-SAIRAS 2005 -The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Flückiger, L., To, V., and Utz, H. (2008). Service-oriented robotic architecture supporting a lunar analog test. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*.

Fong, T., Allan, M., Bouyssounouse, X., Bualat, M. G., Croteau, J., Deans, M. C., Flückiger, L., Lee, S. Y., Lees, D., Keely, L., To, V., and Utz, H. (2008a). Robotic site survey at Haughton crater. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*.

Fong, T., Berka, R., Bualat, M., Diftler, M., Micire, M., Mittman, D., SunSpiral, V., and Provencher, C. (2012). The Human Exploration Telerobotics project. In *Global Space Exploration Conference*, Washington, DC.

Fong, T., Bualat, M., Deans, M., Allan, M., Bouyssounouse, X., Broxton, M., Edwards, L., Elphic, R., Flückiger, L., Frank, J., et al. (2008b). Field testing of utility robots for lunar surface operations. In *AIAA Space*.

Fong, T. W., Bualat, M., Deans, M., Adams, B., Allan, M., Altobelli, M., Bouyssounouse, X., Cohen, T., Fluckiger, L., Garber, J., Palmer, E., Heggy, E., Helper, M., Hodges, K., Hurtado, J., Jurgens, F., Kennedy, T., Kobayashi, L., Landis, R., Lee, P., Lee, S. Y., Lees, D., Lum, J., Lundy, M., Shin, T., Milam, T., Pacis, E., Park, E., Pedersen, L., Schreckenghost, D., Smith, T., To, V., Utz, H., Wheeler, D., and Young, K. (2010). Robotic follow-up for human exploration. In *Space 2010*, pages AIAA–2010–8605. AIAA.

Fong, T. W., Nourbakhsh, I., Kunz, C., Flückiger, L., Ambrose, R., Simmons, R., Schultz, A., and Scholtz, J. (2005). The peer-to-peer Human-Robot interaction project. In *AIAA Space 2005*, Long Beach, California.

Jackson, J. (2007). Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82 –87.

Kroll, P. and Royce, W. (2005). Key principles for business-driven development. *Rational Edge*.

Lee, S. Y., Lees, D., Cohen, T., Allan, M., Deans, M., Morse, T., Park, E., and Smith, T. (2012). Reusable science tools for analog exploration missions: xGDS web tools, VERVE, and Gigapan voyage. *Acta Astronautica*.

Makarenko, A., Brooks, A., and Kaupp, T. (2006). Orca: Components for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*.

Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., and Ingrand, F. (2010). Genom3: Building middleware-independent robotic components. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4627 –4632.

NASA (2012). Smart SPHERES fly high aboard the International Space Station. `http://www.nasa.gov/mission_pages/tdm/telerobotics/spheres.html`.

Nesnas, I., Wright, A., Bajracharya, M., Simmons, R., Estlin, T., and Kim, W. S. (2003). CLARAty: An architecture for reusable robotic software. In *Proceedings SPIE Aerosense Conference*, Orlando, Florida.

Object Management Group (2004). CORBA/IIOP specification. Technical report, OMG, Framingham, MA.

Object Management Group (2012a). Data distribution service (DDS). `http://www.omg.org/spec/DDS`.

Object Management Group (2012b). OMG IDL. http://www.omg.org/gettingstarted/omg_idl.htm.

Object Management Group (2012c). Robotic Technology Component (RTC). http://www.omg.org/spec/RTC.

Orocos (2012). OROCOS: Open Robot Control Software. http://www.orocos.org/.

Pedersen, L., Utz, H., Allan, M., Flückiger, L., Lee, S., and To, V. (2012). Tele-operated lunar rover navigation using LIDAR. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*.

Reeves, G. E. and Snyder, J. F. (2005). An overview of the mars exploration rovers' flight software. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*.

Rowe, S. and Wagner, C. R. (2008). An introduction to the joint architecture for unmanned systems (JAUS). *Ann Arbor*.

RTI (2012). RTI DDS. http://www.rti.com/products/dds/.

Schmidt, D. C., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley & Sons.

Schmidt, D. C. and Vinoski, S. (1999). Programming asynchronous method invocations with CORBA messaging. *C++ Report*, 11(2).

Torres, R., Allan, M., Hirsh, R., and Wallick, M. (2009). RAPID: Collaboration results from three NASA centers in commanding/monitoring lunar assets. In *Aerospace conference, 2009 IEEE*, pages 1 –11.

TwinOaks Computing (2011a). CoreDX DDS Middleware used on-board NASA Lunar Rover. http://www.twinoakscomputing.com/node/346.

TwinOaks Computing (2011b). What can DDS do for you? http://www.twinoakscomputing.com/wp/CoreDX_DDS_Why_Use_DDS.pdf.

Utz, H., Sablatnög, S., Enderle, S., and Kraetzschmar, G. K. (2002). Miro – middleware for mobile robot applications. *IEEE Transactions on Robotics and Automation, Special Issue on Object-Oriented Distributed Control Architectures*, 18(4):493–497.

Venet, A. J. and Lowry, M. R. (2010). Static analysis for software assurance: soundness, scalability and adaptiveness. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*.

Volpe, R. (2003). Rover functional autonomy development for the mars mobile science laboratory. In *Proc. 2003 IEEE Aerospace Conf.*

Wettergreen, D. and Wagner, M. (2012). Developing a framework for reliable autonomous surface mobility. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*.

Willow Garage (2012). Robot Operating System (RoS). http://www.willowgarage.com/pages/software/ros-platform.

ZeroC (2012). Ice manual. http://doc.zeroc.com/display/Ice/Ice+Manual.