FIELD TESTED SERVICE ORIENTED ROBOTIC ARCHITECTURE: CASE STUDY

Lorenzo Flückiger and Hans Utz

Carnegie Mellon University, NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA-94035, USA.
{Lorenzo.Fluckiger,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 NASA Ames Research Center. SORA relies on proven software methods and technologies applied to the robotic world. Based on a Service Oriented Architecture and robust middleware, SORA extends its reach beyond the on-board robot controller and supports the full suite of software tools used during mission scenarios from ground control to remote robotic sites. SORA has been field tested in numerous scenarios of robotic lunar and planetary exploration. The results of these high fidelity experiments are illustrated through concrete examples that have shown the benefits of using SORA as well as its limitations.

Key words: Service Oriented Architecture, Space Robotics, Field Tests Experiments.

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. 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 conducted while allowing IRG's research in human-robot exploration to smoothly evolve. This paper reports on the lessons learned from over six years of experiments with the SORA software system.

1.1. Context

Human-robot exploration of remote locations has been one of IRG's key research topics for more than a decade.

Figure 1: K10 Red Rover in the Arctic, running SORA during the 2007 Haughton Crater Field Test (Canada).

This applied research involves numerous robotics field test experiments conducted to validate the proposed approaches. Most of these field tests take place in remote locations that are good Mars or Moon analogs like shown in Fig. 1. Some of the key requirements for the software system running on IRG's robotic platforms are: 1) enable complex autonomous systems, 2) support a wide range of robots and instruments, 3) permit a variety of exploration scenarios, 4) facilitate the integration with a whole suite of mission tools and 5) allow a dynamic research by a small team. To address these challenging requirements, in 2005 IRG began developing the Service Oriented Robotic Architecture (SORA) which is detailed in [1]. SORA embraces the typical concepts of service oriented systems: encapsulation, communication patterns based on stable interfaces, and reliance on robust middleware. This builds a loosely coupled and highly cohesive system.

The research on software methods and systems for robotics has increased considerably over the past six to eight years. The application of good software practices to handle the complexity of robot systems has lead naturally to the adoption of software architectures well established in computer science. Three of those architectural paradigms are classified in [2]: 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 RTC [3] and RoSta [4], the landscape of solutions is still extremely fragmented. Some approaches are focused on constructing a new type of middleware specifically for robotics like OROCOS [5] or the ROS [6], increasingly adopted in the robotic research community. Other approaches are building component frameworks using an existing middleware like [7] or [8] or are designed to be middleware independent [9].

SORA shares several characteristics common to CBAs and SOAs in robotics and adds the following specificities: 1) SORA extends well beyond the robot controller and is used across the whole mission tool suite and 2) SORA has been used extensively in high fidelity robotic mission simulations.

1.2. Experience with Exploration Robots

In [1] we have shown how SORA has supported a Lunar analog robotic field test during the 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 [10]. Since then, SORA has been used during the summer periods of 2008 at Moses Lake (WA), 2009 at Black Point Lava Flow (AZ) [11] and 2010 at the Haughton Crater site again [12], as well as in a couple of small-scale mission experiments conducted at sites closer to the NASA Ames Research Center. These unique opportunities allowed IRG to test SORA robustness in applied scenarios involving full teams depending on the robotic resource availability (mobility and data gathering). Field experiments also put SORA services interactions (within the robot and to the ground control) in real situations with non homogeneous networks including satellite links with variable Quality of Service (QoS).

1.3. SORA as Mission Backbone

Fig. 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 collocated or distributed across a network, their interactions are based on unified interfaces and are transparently optimized by the supporting middleware.

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.

2. SORA CONCEPTS

It is first important to emphasis that SORA is a *software* architecture supporting robotic systems, and does not define a particular robot *control* architecture [1]. Fig. 3 illustrates a simplified controller constructed with SORA. The details of the services internal structure, and unified services communication modes are described in [1]. The key points concept of SORA are briefly highlighted in this section, starting with the common characteristics shared by SOAs and finishing with the SORA specificities.

2.1. Essential SOA aspects of SORA

This section describes what properties make SORA a typical SOA.

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) [13], 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 at ground control.

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 [14] to implement a publish/subscribe mechanism. This implementation still remains active while SORA transitions to the Data Distribution System (DDS) [15]. DDS is a recent standard by the Object Management Group (OMG) on the publish-subscribe paradigm. At publication, most of the rover telemetry will be also distributed using DDS.

[1] The current control architecture of IRG robots is constructed as a two tiered system combined with some services acting like behaviors, thus it should be probably characterized as a *mixed* control architecture.

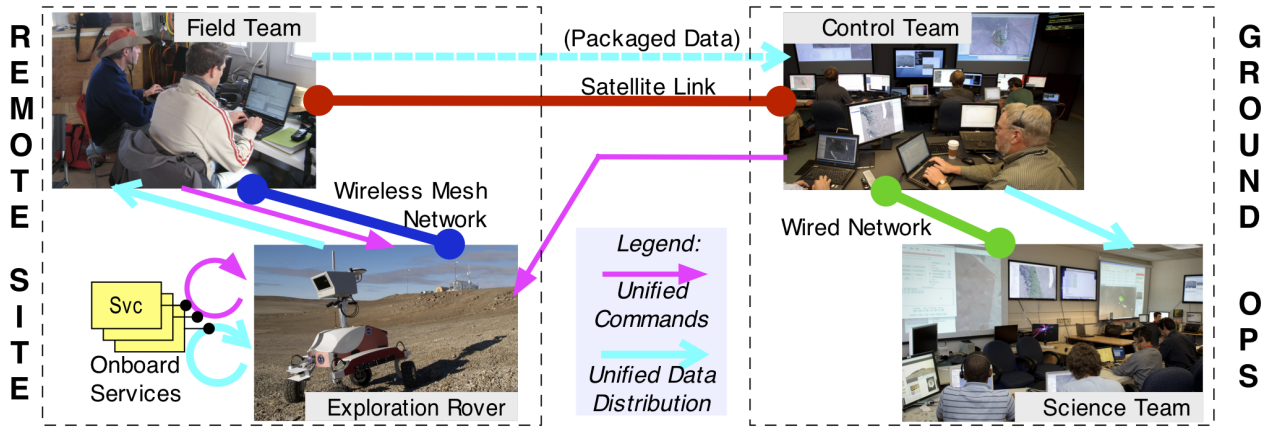


Figure 2: Typical use of SORA across a full field deployment. Common unified interfaces are shared at each node. Data distribution appears identical at all nodes.

2.1.4. Middleware

SORA relies heavily on middleware, specifically the ACE/TAO implementation [16] of the CORBA [14] standard. In addition to CORBA, SORA uses the MIddleware for RObots (Miro) [17]. 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 that will continue to be used for commanding, SORA relies on the RTI [18] DDS implementation for data distribution. DDS offers numerous new Quality of Service (QoS) compared to CORBA for data distribution across heterogeneous, non-reliable networks. For example, building on DDS QoS, IRG was able to conduct a field test where a 50s transmission delay was introduced.

2.2. Specific aspects of SORA

This section describes some characteristics that are unique to 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 scenario. The same configuration mechanism is used also for services not running on the robot, like simulated components. A robot controller assembled for a typical exploration scenario with autonomous navigation and a few science instruments average 45 services. These services could 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.

2.2.2. Loose coupling

Services of a SOA are not subject to the same level of coupling as the components of a CBA. SORA services need to respect some rules, like resource usage to play nice in the overall system. However, these rules are not enforced by a formalism or a compiler. In addition, the loose coupling between services allows freedom on the implementation method of each service. In contrast CBAs can offer facilities for architecture analysis, system composition, and individual components building. IRG is considering using the CORBA Component Model (CCM) [19] in the future to gain some structured CBA benefits, however CCM was not mature enough when SORA was initiated.

2.2.3. Standard messaging between NASA robots

From its inception, SORA exposed a set of messages for data distribution within the robot and to external subscribers. Building on the expertise gained using this system during several years, a collaboration with other NASA centers started an effort to create standard messages for exploration robots across NASA. This effort led to the RAPID project [20]. RAPID defines a set of standard data structures that are shared between the robots of already 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 is Open-Source Software and essentially consists of a set of IDLs plus utilities classes.

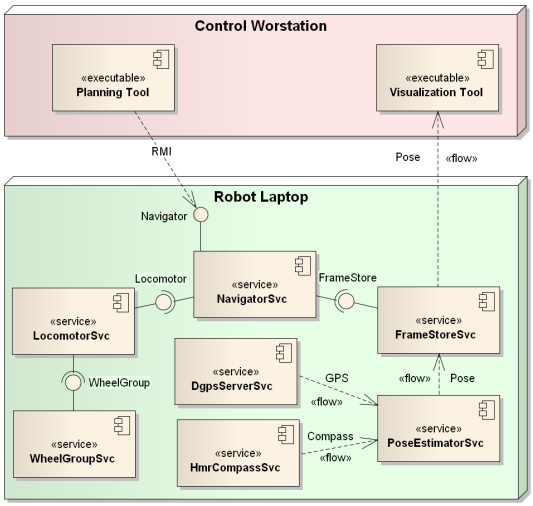


Figure 3: A minimal robot controller built with only a few services. Data distribution is unified across the full deployment. Provided interfaces are transparently accessible within a robot or remotely.

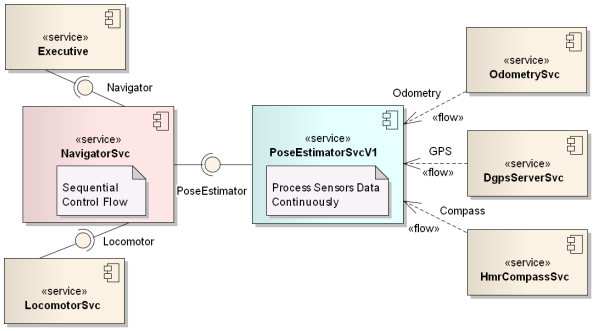
These IDLs are processed to generate code supporting messages that are distributed using DDS.

2.2.4. Validation with field tests

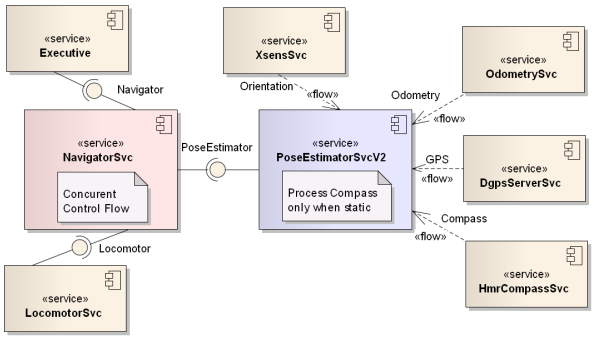
As mentioned above, SORA was deployed in multiple field tests on various rovers. In addition to power the K10 series rovers, SORA has also been tested on a smaller scale rover, K10-mini (footprint of 40cmx30cm), as well as the much larger new IRG rover K-REX (footprint 2mx1.6m). Outside of IRG rovers, SORA also currently supports a navigation system based on a LIDAR for the Centaur-2 rover [21]. The range of situations encountered across these field tests is summarized in Tab. 1. SORA's exposure to these real world situations confirms that SORA meets three key characteristics of robotic space systems: flexibility, scalability and reliability. The unified interfaces across the system, for both collocated and distributed scenarios, provide a great flexibility. Similarly, the facility to create specific robot controllers for particular scenarios by easily assembling different sets of services brings a great flexibility. 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 promote a high level of reliability.

3. SORA BENEFITS

This section describes the benefits of SORA during the IRG robotics field tests. Each of these benefits are generated by architectural concepts and illustrated with a concrete example. Most of the advantages reported below are derived from Service Oriented Architecture specific concepts like encapsulation, communication pattern, and exposition of stable interfaces. In addition, the use of a component configurator pattern and reliance on robust middleware increases the flexibility and reliability of the system.



(a) HMP 2007 Controller



(b) HMP 2010 Controller

Figure 4: Two successive versions of the K10 controller (only few services shown).

3.1. Encapsulation

Encapsulation of robotics capabilities into services exposing well defined interfaces is effectively shielding the overall system from any code modification within services. As long as the IDLs for the interfaces and messages for data distribution remain the same, any change to the internals of a service will not affect other services. This is illustrated in Fig. 4 with the 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. While ignoring the mapping services dependencies for the simplicity of the discussion, we can see that the `Navigator` service has strong dependency on the `Locomotor` service and `PoseEstimator` service. In addition the `Navigator` service is used by the `Executive` service. The navigator has undergone major re-structuring over the years to gain performance and obtain more flexibility. In particular the initial navigator heavily relied the Navigation classes from the CLARAty [22] framework, state of the art at the time. The current navigator replaces the previously sequential model with a newly developed,

Table 1: 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	2Mbps (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, Sofware (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)

concurrent 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 interface.

3.2. 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 while the Publish/Subscribe mechanism is better suited for distributing data to multiple services. 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) [23] pattern which augments the service decoupling and simplifies services implementation. For example, a service can take minutes to complete an operation; however, it can be impractical for the caller of this operation to block its thread of execution while waiting for completion. Using AMI, the call will immediately return and the caller will be notified by a callback when the completion actually occurred. All the complexity of AMI, thread safety, and exception handling is handled by the middleware.

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. In addition, the Publish/Subscribe mechanism decouples the services further as producers of data are totally unaware of the consumers. SORA contains a key service exploiting fully the Publish/Subscribe mechanism: the data logger. This service is a generic data consumer, that can subscribe to any message type and serialize it to file, including a trace of the request/reply pairs of commands. With the Miro *LogPlayer* tool, the log files created can be replayed at one's convenience to analyze a particular situation. The *LogPlayer* also permits data to be fed back to the data-bus, where it can be consumed in the same manner as the original data.

3.3. Stable Interfaces

All SORA interfaces (allowing remote method invocation), and all data structures (participating in the publish/subscribe mechanism) are defined with the IDL language. Each IDL specification is carefully designed to be as generic as possible while allowing access to specific capabilities of the sub-system. 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. Keeping these interfaces stable and their specification in a unique repository (shared by all the parties contributing to software for robotic field tests) permits to easily swap a service for an equivalent one and enables to maintain all the tools around the robot controller up to date.

An example of this evolution is shown in Fig. 4. A new version of the `PoseEstimator` has been written for the second 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 depending on the PoseEstimator did not have to be modified at all.

CORBA interfaces support inheritance. SORA uses this feature extensively to define the services interfaces. In addition SORA extend this polymorphism to the imple-

mentation of the interfaces. Class polymorphism is a powerful concept of any object-oriented language. This is exemplified when used at the service level with interface polymorphism. The best example in SORA are the interfaces to the services encapsulating science instruments functionalities. For each field test IRG rovers carry a new set of science instruments to achieve some particular science goal. Of course, each instrument has a particular set of characteristics that require some specific methods to access its functionalities. However, all these 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 particularly the case for the `Executive` service that executes plans defined by the scientist. A plan contains instructions 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 will command any instrument which inherits and implements that base-interface.

The abstract interfaces in combination with the easy reconfigurability of the controller is also used to replace some (or all) services of the controller for the physical robot with simulated components. The `Locomotor` service on Fig. 3 is responsible for translating high level locomotion commands (translate, drive arc) to individual motor commands regarding the rover kinematics and actuation capabilities. These low level commands are passed to the `WheelGroup` service that abstracts the actual robot hardware. A simulated `WheelGroupSim` service which simulates the robot wheels motion has been developed. It can simply be started in place of the original service to obtain a simulation of the rover motion. Applications like the 3D visualization tool which is used to monitor the rover progress, do not need to be modified at all and it is a matter of switching configuration files to start a real rover controller or a simulated rover controller.

3.4. Component Configurator Pattern

SORA uses the "Component Configurator" pattern [24] to combine the services in a full system. A configuration file specifies which services should be started to create a particular controller. 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 are ranging from a minimal controller containing only the locomotion service to a full blown field test controller requiring a suite of science instruments, passing by controllers containing simulated components. In addition to facilitate developers task, the simplicity and rapidity of creating a new controller configuration also allows tuning controllers regarding 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 (range 30 to 50) running in concert on a robot, while sharing the ressources harmoniously.

3.5. Robust Middleware

The architectural paradigms implemented in SORA would not have achieved such a reliable and extensive set of features without adopting a robust middleware. As mentioned in the Section 2, SORA heavily relies on CORBA coupled Miro, and increasingly on DDS. These dependencies obviously are imposing some constraints due to the choice of a specific middleware (see Section 4 for the drawbacks). Middleware is pervasive, so replacing any middleware for another one would require 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. The ACE/TAO CORBA implementation went though several release cycles, so about once a year SORA updates to use the latest revision. Each new revision of ACE/TAO resolves some issues and bring new improvements. SORA directly benefited from these new versions representing a considerable amount of work from outside parties. At the same time, CORBA being a standard, new revisions of the implementation only requires minor changes on the user side. The same stability argument can be made for Miro. The few changes in the Miro source code over the years is a praise to its reliability and SORA has been very well supported by Miro's initial set of features.

Finally, appropriate usage of middleware frees the developers of the robotic software from issues or changes of the lower layers. For example, as shown in Fig. 2, the Field Site and Ground Operations are connected using an unreliable satellite link. To cope with lower and intermittent data rates, the robot telemetry is 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 Field Site or Ground Operations. The exact same applications are running on each site, not cognizant if the connection is supported by a direct optical fiber link (situation for the local test site at NASA Ames) or by a satellite link.

4. SORA SHORTCOMINGS

The long-term use, continuous development and intensive field testing of SORA provided a good stress-test uncovering weaknesses in the design and implementation choices as well as the deployed software technologies of the SORA architecture. In this section we want to discuss some of those: scalability of the publish/subscribe mechanism, re-use of data structures, synchronization of services and middleware acceptance by external parties.

4.1. Publish/Subscribe Scalability

The CORBA Notification service is designed as a central monolithic data-relay. This obviously makes it a

single-point of failure and is not scalable to complex distributed applications. Also, the QoS options do not support bandwidth-management very well, which is a major scalability concern in heterogeneous networks. Furthermore the Notification service has other short-comings, such as a very inefficient transfer of type-code information, that are of concern especially on low-bandwidth network-links. We overcame these problems by deploying a customized extension that allows to create a federation of notification service instances exchanging events between instances on a separate data-channel [25]. We also added a time-based filter, implementing message-frequency limits between nodes of the federation.

While this system provides a solution for our specific requirements it is obviously not a generalized system aimed at addressing all design deficiencies of the Notification service. SORA transition to DDS allows to overcome these limitations regarding data distribution. DDS extensive QoS capabilities permits its deployment in very difficult network environments. Furthermore it supports different modes of data-dissemination that allow implementation of a wider set of communication patterns over a data-bus. On the down-side, adding a second middleware package neither helps the footprint nor the complexity of the software architecture.

4.2. Rigidity of Data Structures

The publish-subscribe model of data-distribution is central to many SORA like distributed systems. Unfortunately this model violates some of the abstraction concepts of the object-oriented paradigm. 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 subscribing to a generalized concept (i.e. position), to ignore the specific sensor information (i.e. additional GPS data fields). However, the CORBA Notification service as well as DDS only allow retrieval of the payload-type of an event that was put in on the publisher side.

In consequence, SORA data producers (like 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 type-signatures.

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, too. Triggering activity in one service on an event emitted by another service is straight forward, but other synchronization primitives do not exist.

This becomes more of an issue, with simulation, when single-stepping and faster-than-real-time execution become of interest. Synchronization of the systems real-time clock 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.

4.4. Middleware Acceptance

Objective criteria generally state a clear benefit of middleware over ad-hoc solutions for distributed systems development. Nevertheless, the acceptance of 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 foot-print as other frameworks and libraries that are regularly used in the development of large-scale systems (GUI toolkits, databases, JIT-compiler etc). It is difficult to over-come established misconceptions regarding the complexity and performance of middleware In our experience, the complexity of middleware can be managed by a small number of domain experts, though. The other developers then do not have to be concerned about the details of the distributedness of their applications.

One factor which stays true 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, the poor readability of the generated code also makes it difficult for the non-domain expert to directly look up the method signatures in the code generated by the IDL-compiler.

5. CONCLUSION AND FUTURE WORK

This paper describes the Service Oriented Robotic Architecture (SORA) design concepts, 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 term of flexibility, scalability and re-

liability. At the same time, these experiments helped to identify SORA limitations in those areas.

Future work on SORA includes refining some of the SOA concepts, fully replacing the CORBA publish/subscribe mechanism with the DDS standard, and continuing to standardize the robotic interfaces with other NASA centers. In addition, the SORA source code has been cleared for release under the NASA Open Source Agreement licensing and thus will be available to a larger community for evaluation and contributions.

The advantages of SORA extend beyond the domain of the robot controller architecture. SORA is the backbone supporting IRG field test scenarios by connecting the various robotic mission tools with a powerful distributed system infrastructure. The SORA design and implementation has enabled a full eco-system of robotic capabilities, and will continue to smoothly support their evolution in the future.

ACKNOWLEDGMENTS

This work was supported by the NASA Exploration Technology Development Program and the NASA Enabling Technology Development and Demonstration Program The authors would like to thanks all the individuals who contributed to SORA: Mark B. Allan, Xavier Bouyssounouse, Matthew Deans, Susan Lee, Mike Lundy, Eric Park, Liam Pedersen and Vinh To.

REFERENCES

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

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

[3] Object Management Group. Robotic Technology Component (RTC). `http://www.omg.org/spec/RTC`, 2012.

[4] RoSta. Robot standards and references architectures. `http://www.robot-standards.org/`, 2009.

[5] Orocos. OROCOS: open robot control software. `http://www.orocos.org/`, 2012.

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

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

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

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

[10] Fong, T., Allan, M., Bouyssounouse, X., et al. Robotic site survey at haughton crater. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*, 2008.

[11] Deans, M. C., Fong, T., Allan, M., et al. Robotic scouting for human exploration. In *AIAA Space 2009*, Pasadena, California, September 2009.

[12] Fong, T. W., Bualat, M., Deans, M., et al. Robotic follow-up for human exploration. In *Space 2010*, pages AIAA–2010–8605. AIAA, September 2010.

[13] Object Management Group. OMG IDL. `http://www.omg.org/gettingstarted/omg_idl.htm`, 2012.

[14] Object Management Group. CORBA/IIOP specification. Technical report, OMG, Framingham, MA, April 2004.

[15] Object Management Group. Data distribution service (DDS). `http://www.omg.org/spec/DDS`, 2012.

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

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

[18] RTI. RTI DDS. `http://www.rti.com/products/dds/`, 2012.

[19] Schmidt, D. C. & Vinoski, S. Object interconnections: The CORBA component model: Part 1, evolving towards component middleware. *C/C++ Users Journal*, February 2004.

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

[21] Pedersen, L., Utz, H., Allan, M., et al. Tele-operated lunar rover navigation using LIDAR. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS)*, 2012.

[22] Nesnas, I., Wright, A., Bajracharya, M., et al. CLARAty: An architecture for reusable robotic software. In *Proceedings SPIE Aerosense Conference*, Orlando, Florida, April 2003.

[23] Schmidt, D. C. & Vinoski, S. Programming asynchronous method invocations with CORBA messaging. *C++ Report*, 11(2), February 1999.

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

[25] Hirsh, R. L., Simon, C. L., Tyree, K. S., et al. Astronaut Interface Device (AID). In *Proceedings of AIAA Space 2008*, San Diego, CA, September 2008. AIAA.