

First-Class Protocols for Agent-Based Coordination of Scientific Instruments

Tim Miller, Peter McBurney
Department of Computer Science,
University of Liverpool, UK
{tim, p.j.mcburney}@csc.liv.ac.uk

Jarred McGinnis, Kostas Stathis
Department of Computer Science,
Royal Holloway, University of London, UK
{jarred, kostas}@cs.rhul.ac.uk

Abstract

*The coordination of distributed processing is of great interest to a number of research communities. However these research communities, such as those involved in e-science GRIDs and multi-agent systems, view this problem from disparate viewpoints. This paper aims to contribute to the necessary reconciliation of these perspectives. By demonstrating the coordination of processes whether reactive (such as web services) or proactive (such as autonomous agents) can be done with a single representation using a protocol language, *R.A.S.A.* The multi-agent paradigm introduces, into any model of distributed systems, flexibility and autonomy that can be daunting and intimidating to scientists accustomed to more orthodox approaches. It is for this reason that it is important that the model for coordination of this system is reliable, verifiable, inspectable, referable, composable and executable. The language *R.A.S.A.* provides this functionality, which we extend its use for not only agent interaction protocols but also to express workflows. The language for expression then becomes the domain of discourse as well as potentially the language for the workflow's execution.*

Keywords: interaction protocols, web services, multi-agent systems, coordination.

1 Introduction

The rise of the Internet and the increasing use of computers in scientific research has led to the development of scientific GRIDs and e-science systems. In these systems, scientific instruments and computational resources held in multiple physical locations are connected together via the Internet, and made accessible to researchers in yet other locations. Thus, for example, a scientist in location A may submit samples for scanning to a sophisticated electron microscope in location B, the output of which is subsequently transferred to a computer in location C for rendering as an image and presentation back to the scientist at A [2]. De-

pending on the resulting output, the scientist may decide further scanning and analysis of the sample or the output data is needed, and so may request additional actions be undertaken by the microscope at B, the image processor at C, and/or other computer analysis processors at a location D.

In such scientific workflows, some instruments or resources may be directly under the control of the scientist undertaking the work, and so he or she does not need to negotiate for their use. More typically, however, usage of the instruments or resources will need to be negotiated with the resource-owner by the scientist or by some agent representing him or her. If the complete sequence of measurement, analysis and presentation activities is known in advance, a detailed workflow can readily be developed for each such scientific task, and negotiation for appropriate resource access undertaken in advance of commencement of the workflow. However, for many complex scientific procedures, later tasks may depend on the outcomes of earlier tasks, with even the instruments or resources to be used only specifiable once these earlier outcomes become known. In such situations, scientists need to be able to develop workflows at the time of task execution (or, as computer scientists would say, at *runtime*) and to be able to modify such workflows dynamically, as task execution proceeds; modifications would arise as a consequence of the outcomes of earlier actions and/or the changing availability of needed resources.

In this paper, we present a formal language to enable scientists, and/or software agents acting on their behalf, to reason about, select and modify workflows, and to do so at run-time. The language treats interaction protocols as first-class entities, so that they can be referenced, inspected, composed, and invoked by the agents engaged in interaction. A particular feature of this approach is that resources deployed as part of a workflow are referenced in a uniform fashion, regardless of the extent to which a particular agent owns or controls them. Thus, a workflow may refer both to resources which are owned by the scientist, where no negotiation may be needed to use the instrument, and to resources owned by others, where such negotiation may be

necessary. This feature means that reactive processes, such as web-services, and autonomous processes, such as agents representing other resource owners, are both incorporated into the framework.

The protocol representation language, *R.A.S.A.*, presented here was introduced in [4]. That paper has a description of related work on defining process algebras and languages for agent interaction protocols. A key advantage of the *R.A.S.A.* framework over other frameworks is that *R.A.S.A.* has the same syntax and semantics at all dialogue levels. Thus, individual utterances, sequences of utterances, protocols, and combinations of protocols can all be reasoned-over, modified, composed and invoked by agents participating in an interaction using the same reasoning mechanism.

The key contribution of this paper is to demonstrate how the *R.A.S.A.* framework can be used for dynamic specification and execution of scientific workflows, where such workflows may involve use of both reactive and autonomous processes. Section 2 presents the syntax of the *R.A.S.A.* language, while Section 3 discusses its application to scientific domains. An example of the application of *R.A.S.A.* to a workflow involving an electron microscope is presented in Section 4. Section 5 concludes the paper.

2 The *R.A.S.A.* Language

The *R.A.S.A.* language, part of the larger *R.A.S.A.* framework, combines constraints and process algebra to model interaction protocols as *first-class* entities. By *first-class*, we mean that the protocols exist as documents within a system, rather than as abstractions emerging from the behaviour of participating entities. Using a constraint language, the *rules* – that is, the conditions that must hold for particular messages to be sent; and the *effects* — that is, the effect that sending a message has on the system; are modelled such that participants in a protocol can calculate whether the rules allow certain messages to be sent, and the effect that sending a message has. Such an approach allows participants to reason, at runtime, about their course of interaction using constraint solving techniques. Not only does such an approach decouple the entities from their protocols — something which is unanimously encouraged in software engineering —, but it allows participants to learn about new protocols, and to compose protocols at runtime to bring about more complex interactions. This would result in systems in which, for example, an agent has a library of interaction protocols, and searches through this library at runtime to decide which protocol best suits its current goals. If no single protocol is suitable for the agent, composition of these may offer an alternative.

2.1 Syntax

The *R.A.S.A.* protocol language resembles that of many process algebras. However, we specify the rules and effects of protocols using an underlying constraint language. Such specification allows agents equipped with the necessary tools to reason about this language to determine when rules are satisfied, to calculate the effect of sending a particular message, and to devise strategies for interaction at runtime.

A *R.A.S.A.* protocol is an annotated tree of interactions between entities, in which an interaction is a sequence of transitions between states. An *annotation* is a triplet of constraints, in which the first constraint represents the precondition that must hold for a transition to occur, the second constraint represents the message to be sent, and the third constraint is a postcondition, which represents the constraints that holds after an enabled transition occurs. Branches in the tree represent choices to be made by one or more participants.

Let ϕ represent constraints defined in the constraint language, c communication channels, N protocol names, and x a sequence of variables. Protocol definitions adhere to the following grammar.

$$\pi ::= \epsilon \mid \phi \xrightarrow{c(i,j).\phi} \phi \mid \pi; \pi \mid \pi \cup \pi \mid N(x) \mid \mathbf{var}_x^\phi \cdot \pi$$

We use π as a meta-variable to refer to protocols; subscripts and superscripts are used to denote distinct meta-variables. ϵ represents the empty protocol, in which no message is sent and there is no change to the protocol state. A protocol of the format $\phi \xrightarrow{c(i,j).\phi_m} \phi'$ is an atomic protocol. It represents that i can send the constraint ϕ_m to j over channel c only if ϕ holds in the current state, in which i and j are values in the constraint language. After the message is sent, the new state of the protocol is updated using ϕ' . We use this to specify rules and effects of protocols: the precondition represents a rule for a protocol because ϕ_m can only be sent if this precondition is true; and the postcondition represents the effect that sending ϕ_m has on the state.

The protocol $\pi_1; \pi_2$ denotes the sequential composition of two protocols, such that all of protocol π_1 is executed, then protocol π_2 . The protocol $\pi_1 \cup \pi_2$ denotes a choice of two protocols. $N(x)$ denotes a reference to a protocol π named $N(y)$, with variables y renamed to x , such that any occurrence of N is equivalent to its definition, π . The protocol $\mathbf{var}_x^\phi \cdot \pi$ denotes the declaration of a local variable x , with the constraints ϕ on x . The scope of x is limited to the protocol π , and the constraints on x do not change on x throughout its scope; that is, x is a constant.

Let N be a name, y be a sequence of variable names, and π be a protocol. A *protocol specification* is defined as a set of definitions of the form:

$$N(y) \cong \pi$$

For a detailed description of the \mathcal{RASA} protocol language and its semantics, see [4].

3 Application for Scientific Instruments

The e-science and GRID research is a domain in which the use of a first class protocol language such as \mathcal{RASA} is useful, especially if the requirements of the scientific instruments, the processing services and users include complex and context dependent coordination. By expressing the coordination language used in such a way to allow inspection, verification, execution, composition and transfer, the parties involved have the autonomy and capability to confer upon and produce a commonly acceptable solution to problems at run time. This section will focus on the general attributes of GRID enabled scientific instruments. Though each category could gain advantages by employing a first-class protocol representation for their coordination, there are certain categories that present problems for which a language like \mathcal{RASA} would be essential.

3.1 A categorisation of scientific instrument coordination

A possible categorisation along two axes, the role relationship of the agents and whether they collaborate upon the creation of the workflow, is shown in Table 1.

Peer-to-peer (non-collaborative): Agents are attached to an instrument and could be either a provider or requester. They have access to in-house web services and processing but there are situations that require the agent to delegate or solicit other agents, who have their own instrument and services, to provide the solution. The agents do not collaborate because the agent is only interested in results not attainable on its own. For example the agent needs to corroborate a result by having a third party run a similar process on a data set. A discussion of the workflow is not needed because it is irrelevant or trivial. The dialogue between the agents can be a contract-net protocol [3] to find the agent who can provide the requested service or if need be a more complicated negotiation protocol that allows for counter proposals and proposal refinement.

Requester/provider (non-collaborative): A scientist or an agent makes a request for a product and the instrument's agent is charged with developing a workflow to provide that product. The agents never switch roles; one is always a requester and the other is always a provider. The requester has no knowledge or concern about the process required to produce the product. Instead, it knows that there are provider agents that should be able to provide the product.

The agents discuss what results are possible but the workflow will be made on the instrument side. The protocol language used to describe the interaction between the agents can be used for the reasoning about which workflows are to be executed. Since the protocols have properties, there would be a mapping between the product requested to the actual procedure it takes to produce it. The interaction protocol for the agents can also be a contract-net or negotiation protocol.

Peer-to-peer (collaborative): Rather than simply delegating tasks to each other, it is possible to imagine instruments that could collaborate to solve problems and process data that each know could not be done in isolation. The topic of the dialogue between the instruments' agents shifts from one of *what* to *how*. Each agent is interested in how it is possible to achieve its goals and not just what the end product will look like. Different instruments will have access to different services and processing power. The agents would then deliberate on the optimal division of labour and formulating the workflow to be executed. This category would be the best served by the ability to reason about the properties of protocols and use those to compose protocols for later execution. For example, agent A suggests starting with service 34 because of property β holding and agent V agrees and suggests doing service 15 because its input is the output of web service 34. The agents agree and can then compose a protocol with this order of execution confident that the properties will hold.

Requester/provider (collaborative): In this category the roles of agents are fixed. The requester itself does not have any services to offer. Instead it acts as coordinator who knows how to compose the workflow amongst a group of providers. Some of its requests will be impossible or the offered services will not be adequate. The dialogue between the requesters and providers becomes one of deliberation as they attempt to come to an commonly agreeable workflow to be executed. Similar to the peer-to-peer category, agents will need to be able to compose a workflow that hold properties that are commonly acceptable. To do so, a representation of the workflow as a \mathcal{RASA} protocol is useful.

Table 2 summarises the interaction protocols that would be needed for the four categories. It becomes apparent that it is the complexity from collaborative dialogues that requires the functionality of being able to refer and reason about the coordination mechanism. The representation of workflows as \mathcal{RASA} protocols provides this functionality. However non-collaborative dialogues could still make use and take advantage of first class protocol representations, enabling agents to consider their function and requirements within an interaction protocol.

Table 1. Categories of agent-led scientific instruments

	peer to peer	requester / provider roles
non-collaborative	1 end result only concern workflow is not discussed different roles can be adopted	2 end result only concern workflow is not discussed roles are static
collaborative	3 process also a concern agents coordinate amongst themselves	4 process also a concern requester acts as a coordinator

Table 2. Dialogues and protocols for the categories

non-collaborative	contract-net negotiation workflows composed privately
collaborative	deliberation multi-levelled dialogues workflows are composed as first-class protocols by the dialogues

3.2 *RASA* for collaborative microscope MAS and WF execution

The authors of [1] describe a scenario that fits into category 1 (non-collaborative and peer-to-peer) from Table 1. This work also uses an interaction protocol language to model the ordering of processes of a workflow intermingled with the social norms for the agents. A contract net protocol is used to obtain results from other agent enabled telescopes to address anomalies discovered from local computation. The benefits this approach could gain by a more formal language such as *RASA* have already been alluded to in previous sections. However it is the work described in [2], being categorised as a collaborative scenario that is of the most interest. The requester could be an agent mediating human requests or autonomous. This agent has a sample that it needs to be analysed by an electron microscope at another location. These expensive and complicated machines can only be operated by someone with training. It is feasible that an agent attached to the instrument has a knowledge representation of this training and knows the capabilities and restrictions of the equipment. The requesting agent may have a plan (a workflow) of activities it wants to perform as it knows the basic operations of the machine (e.g. scan, move, zoom, etc). However there may be sequences and combinations that the instrument's agent cannot allow. It is this scenario where the agents, by having a representation of the workflow as a *RASA* protocol, can plan and deliberate on a course of action that will satisfy the requesters requirements without contravening the provider's restrictions. The well-defined rules of composition in *RASA* provide good

support for generating these courses of action at runtime.

4 An Example using *RASA* collaborative scientific workflow

Collaboration, being concerned not only with the end results but the process by which those results are obtained, requires the ability for agents to understand the interaction protocols and workflows they intend to use. Static and brittle coordination models will not be able to cope with the problems that arise. A protocol language like *RASA* can not only express those models in a formal manner but allow reasoned, verifiable and dynamic modifications to the model while in use. This section will describe, in practice, what this process entails by describing a collaborative scenario involving the remote use of an electron microscope. The protocols described allow the agent enabled instrument and user to confer upon a course of action by composing a workflow also represented as a *RASA* protocol.

4.1 Negotiating Workflows

We assume the following basic ontological definitions:

- $isWf(Wf)$ is true if and only if Wf specifies a valid workflow; that is, a *RASA* specification.
- $prop(P, Wf)$ states that agent P has proposed the workflow Wf , in which $isWf(Wf)$
- $cmt(P, Wf)$ commits the agent P to workflow Wf , in which $isWf(Wf)$. A party may be committed to a par-

ticular workflow even though they have to do nothing to fulfil that commitment.

The protocol progresses in several stages. First, the protocol is opened by an agent requesting that another agent open a dialogue. The receiving agent can choose to withdraw, or can enter the dialogue, at which point the agents begin to negotiate. The variable ϕ is a constraint in the underlying communication language, and it represents the state of affairs that the requester wishes to achieve with the workflow.

$$\text{Protocol}(Pi, Pj, \phi) \hat{=} \text{Open}; (\text{Enter}; \text{Neg} \cup \text{Withdraw})$$

The open, enter, and withdraw dialogues are specified as follows:

$$\text{Open}(Pi, Pj, \phi) \hat{=} Pi \neq Pj \xrightarrow{c(Pi, Pj).open_dialogue(\phi)} \top$$

$$\text{Enter}(Pi, Pj, \phi) \hat{=} Pi \neq Pj \xrightarrow{c(Pj, Pi).enter_dialogue(\phi)} \top$$

$$\text{Withdraw}(Pi, Pj, \phi) \hat{=} Pi \neq Pj \xrightarrow{c(Pj, Pi).withdraw(\phi)} \top$$

The negotiation artifacts are workflows. The *Neg* sub-protocol consists of declaring *Wf*, the workflow that is to be negotiated next, and either proposing this, in the case that it has not been proposed before, or express a preference for this over the previous proposals, followed by an evaluation phase.

$$\text{Neg}(Pi, Pj) \hat{=} \mathbf{var}_{Wf}^{isWf(Wf)}. ((\text{Prop} \cup \text{Pref}); \text{Eval})$$

An agent can only propose a workflow that has not been previously proposed, as outlined by the precondition:

$$\text{Prop}(Pi, Pj, Wf) \hat{=} \neg \text{prop}(Pi, Wf) \xrightarrow{c(Pi, Pj).prop(Wf)} \text{prop}(Pi, Wf)$$

An agent can only express a preference for a workflow that has been previously proposed, as outlined by the precondition:

$$\text{Pref}(Pi, Pj, Wf) \hat{=} \text{prop}(Pi, Wf) \xrightarrow{c(Pi, Pj).pref(Wf)} \top$$

An agent can accept the last proposal made by the other negotiating agent. If this is the case, both agents are committed to the workflow (although one of the agents may not actually need to perform any action to fulfil their commitments).

$$\text{Accept}(Pi, Pj, Wf) \hat{=} \text{prop}(Pj, Wf) \xrightarrow{c(Pi, Pj).accept(Wf)} \text{cmt}(Pi, Wf) \wedge \text{cmt}(Pj, Wf)$$

If the most recent proposal is accepted, the negotiation dialogue is closed, and the workflow agreed between the two parties is appended to the current protocol for execution. If the most recent proposal is rejected, the negotiation stage is iterated, but with the roles reversed; that is, the agent proposing or expressing a preference for the previous proposal will receive a proposal or preference from the other.

$$\text{Eval}(Pi, Pj, Wf) \hat{=} \text{Accept}; \text{Close}; Wf \cup \text{Neg}(Pj, Pi)$$

$$\text{Close}(Pi, Pj, Wf) \hat{=}$$

$$\text{cmt}(Pi, Wf) \wedge \text{cmt}(Pj, Wf) \xrightarrow{c(Pi, Pj).withdraw} \top$$

4.2 Workflows

In this section, we outline and specify an example workflow from the microscopy scenario. In this workflow, an agent representing the microscope requests that the microscope moves to a set of coordinates, requests that the microscope takes an image of these coordinates, and then returns the image. This image is then returned to party who had initially requested the image.

We assume that the microscope understands the following commands:

- *move*(*X*, *Y*): move the microscope to point to the coordinates (*X*, *Y*).
- *scan*: scan the image at the current coordinates of the microscope.

We also assume the following ontological definitions:

- *isController*(*C*, *M*), which is a relation that holds if and only if *C* is the controller of the device *M*.
- *isCoord*(*X*, *Y*), which is a relation that holds if and only if (*X*, *Y*) is a valid coordinate.

The outline of this protocol is below. In this example, *C* is the agent representing the microscope (the controller), *R* is the agent that has requested the image, *M* is the microscope, and *X*, *Y* are the coordinates for the image. *RequestImage* is the top-level of the specification. In this protocol, the microscope is moved to position, requested to take an image, and the image is sent to the original requester.

$$\text{RequestImage}(C, R, M, X, Y) \hat{=}$$

$$\text{Move}(C, M, X, Y); (\text{Take}(C, R, M) \cup \text{Fail}(C, R, M))$$

The *Move* protocol consists of the controller sending the message *request*(*move*(*X*, *Y*)) to its microscope, requesting that the microscope moves to the specified coordinates.

$$\text{Move}(C, M, X, Y) \hat{=}$$

$$\text{isController}(C, M) \xrightarrow{c(C, M).request(move(X, Y))} \top$$

Should the microscope be incapable of moving to the coordinates, due to mechanical failure for example, then it will return the message *failure*(*move*(*X*, *Y*)). Our assumption that the microscope is not autonomous means that this will only happen in the case that it cannot move to the coordinates — never if it can.

$$\text{Fail}(C, R, X, Y) \hat{=} \top \xrightarrow{c(M, C).failure(move(X, Y))} \top; \top \xrightarrow{c(C, R).failure(RequestImage(C, R, M, X, Y))} \top$$

If the microscope moves to the coordinates without problem, that it will confirm the move with its agent; the post-condition being $at(M, X, Y)$, representing that the microscope is at the coordinates (X, Y) .

$$TakeImage(C, R, M, X, Y) \hat{=} \\ Confirm(C, M, X, Y); Scan(C, M); Send(C, R)$$

$$Confirm(C, M, X, Y) \hat{=} \\ \top \frac{c(M, C).confirm(move(X, Y))}{\top} at(M, X, Y)$$

The *Scan* protocol is executed only in the case that the microscope could move to the specified coordinates. The controller sends a request to its microscope that it take an image of the current coordinates, using the command *scan*. For simplicity, we assume that this cannot fail, but a failure condition could be handled in the same way as failure to move to coordinates. Therefore, the only response to the microscope is to get the image at the coordinates (X, Y) , and return this image to its controller.

$$Scan(C, M) \hat{=} \\ \text{var}_{X, Y} \frac{at(M, X, Y) \cdot (isCoord(X, Y) \frac{c(C, M).request(scan)}{\top})}{\top}; \\ \text{var}_I \frac{image(X, Y, I)}{\top} \\ \top \frac{c(M, C).inform(image(X, Y, I))}{\top} image(X, Y, I)$$

Finally, the controller sends to the image data to the agent that submitted the original request.

$$Send(C, R) \hat{=} \\ \text{var}_{X, Y, I} \frac{image(X, Y, I)}{\top} \frac{c(C, M).inform(image(X, Y, I))}{\top} \top$$

Specifying this workflow as a proposal would involve proposing the name, as well as instantiations of the parameters C, R, M, X , and Y . Therefore, using the negotiation protocol outlined in Section 4.1, a participant could send the following message:

$$c(i, j).prop(RequestImage(j, m, i, 10, 10))$$

This indicates that agent i would like an image of the coordinates (X, Y) to be taken by microscope m , controlled by the agent j . Agent j would (hopefully) be capable of reasoning about this workflow, and should it think this reasonable and possible, it may accept this protocol, committing to perform the actions in it. After the *Close* protocol in Section 4.1, the protocol *RequestImage(j, m, i, 10, 10)* would be the next part to execute. Upon receiving the image, the requester may want to propose another workflow, for example, the requester may see something on the image that it would like investigate, so it negotiates with the controller to zoom in on the previous coordinates and scan another image. Allowing this dynamic composition is essential to the operation of the microscope, because the actions that the requester would like to execute on the microscope would often depend on the results of previous workflows. While other languages could be used to express these workflows,

we think that *RASA* is suitable, and that its familiar syntax and semantics make composition straightforward.

5 Conclusions

In this paper, we have presented a formal language and framework for multi-agent co-ordination of scientific instruments in complex scientific workflows. The language permits humans or software agents to reference, reason-over, compose, invoke and modify protocols for interaction and co-ordination in real-time, so that workflows may be modified during execution. The language treats protocols as first-class entities and has a uniform syntax and semantics at all levels of interaction, whether discrete utterances, sequences of utterances, protocols, or combinations of protocols. In addition, the language enables uniform representation within a workflow both of reactive processes, such as web-services, and of autonomous processes, such as scientific instruments or computational resources controlled by others. Protocol composability means that any subsidiary inter-agent negotiations needed to make use of a specific scientific instrument may be included in the workflow along with the workflow actions which precede or may follow the use of the instrument. In future work, we plan to implement software agents able to interact using the *RASA* language in order to assess the applicability of these proposals for automated interactions.

Acknowledgements

We are grateful for financial support from the EC ARGU-GRID project (ArguGRID-IST-035200), website: <http://www.argugrid.org>, the EC PIPS project (EC-FP6-IST-507019), website: <http://www.pips.eu.org>, and the EC ASPIC project (IST-FPC-002307), website: <http://www.argumentation.org/>.

References

- [1] A. Barker and R. G. Mann. Flexible service composition. In *Cooperative Information Agents*, volume 4149 of *Lecture Notes in Computer Science*, pages 446–460. Springer, 2006.
- [2] W. Costello, A. Bleloch, P. Goodhew, P. McBurney, and R. Paton. A new inclusive approach to remote microscopy (extended abstract). In *Proceedings of the Thirteenth European Microscopy Congress (EMC 2004)*, Antwerp, Belgium, 2004.
- [3] FIPA. Fipa contract net interaction protocol specification, 2002.
- [4] T. Miller and P. McBurney. Towards a lightweight formal language for first-class agent interaction protocols. In G. O’Hare, A. Ricci, M. O’Grady, and O. Dikenelli, editors, *Proceedings of Engineering Societies in the Agents World*, 2006.