nDrites: Enabling Laboratory Resource Multi-Agent Systems

Katie Atkinson[1], Frans Coenen[1], Phil Goddard[2], Terry R. Payne[1], and Luke Riley[1,2]

(1) Department of Computer Science, University of Liverpool,
Liverpool L69 3BX, United Kingdom, {atkinson,coenen,payne,l.j.riley}@liverpool.ac.uk.
(2) CSols Ltd., The Heath, Business & Technical Park, Blacon,
Runcorn WA7 4QX, United Kingdom, {phil.goddard,luke.riley}@csols.com.

Abstract. The notion of the multi-agent interconnected scientific laboratory has long appealed to scientists and laboratory managers alike. However, the challenge has been the nature of the laboratory resources to be interconnected, which typically do not feature any kind of agent capability. The solution presented in this paper is that of nDrites, smart agent enablers that are integrated with laboratory resources. The unique feature of nDrites, other than that they are shipped with individual instrument types, is that they possess a generic interface at the "agent end" (with a bespoke interface at the "resource end"). As such, nDrites enable the required interconnectivity for a Laboratory Resource Multi Agent Systems (LR-MAS). The nDrite concept is both formally defined and illustrated using two case studies, that of analytical monitoring and instrument failure prediction.

1 Introduction

Analytical laboratories form a substantial industry segment directed at chemical analysis of all kinds (clinical, environmental, chemical, pharmaceutical, water, food etc). Supplying this marketplace is a \$100B per annum industry. Laboratory instruments come in many forms but are broadly designed to undertake a particular type of chemical analysis. Examples of laboratory instrument types include: inductively coupled plasma - mass spectrometers (ICP-MS) for elemental analysis and Chromatography systems for analyte separation. Such laboratory instruments, although usually "front-ended" by a computer resource of some kind, typically operate in isolation. This is because the interfaces used are specific to individual instrument types (of which there are thousands) and individual manufactures. The industry acknowledges that there are significant benefits to be gained if instruments, of all kinds, could "talk" to each other and to other devices [10, 22]; an ability to support remote monitoring/managing of instruments would on its own be of significant benefit. A potential solution is the adoption of a Multi-Agent Systems (MAS) approach to laboratory resource interconnectivity: a Laboratory Resource Multi Agent System (LR-MAS).

However, at present, there is no simple way whereby the LR-MAS vision can be realised. This is not only because of the multiplicity of different interfaces for different models, but also the complex mappings, translations and manipulations that have to be undertaken in order to achieve the desired interconnectivity. Even when just considering specific laboratory instruments, rather than the wider range of laboratory resources, there are many thousands of models being sold at any one time and a huge variety of legacy systems still in routine use. The limited connectivity that exists is largely focused on what are known as Laboratory Instrument Management Systems (LIMS); systems that receive and store data from instruments (for later transmission to laboratory clients) and manage wider laboratory activities. Some software does exist to facilitate connectivity, for example the L4L (Links for LIMS) software package produced by CSols Ltd[1] (a provider of analytical laboratory instrument software); but this still requires expensive on-site visits by specialist engineers to determine the desired functionality and the nature of the bespoke interfacing. All this serves to prevent the adoption of MAS capabilities within the analytical laboratory industry, despite the general acknowledgement that large scale MAS connectivity will bring many desirable benefits [10, 22].

The technical solution presented here is that of "smart agent enablers" called *nDrites*; an idea developed as part of a collaboration between CSols Ltd and a research team at the University of Liverpool, directed at finding a solution to allow the realisation of the LR-MAS vision. The nDrite concept is illustrated in Figure 1. As shown in the figure, nDrites interact, at the "resource end", in whatever specific way is required by the laboratory resource type in question; whilst at the other end nDrites provide generic interaction. Note that in the figure, for ease of understanding, the nDrite is shown as being separated from the laboratory resource (also in Figure 2), in practice however nDrites are integrated with laboratory resources. Thus nDrites provide system wide communications so as to allow agents to interact with laboratory resources to (say): (i) determine the current state of an entire laboratory system, (ii) determine all past states of the system (system history) or (iii) exert control on the laboratory resources operating within a given laboratory framework. Thus, in general terms, nDrites are a form of intelligent middleware that facilitate LR-MAS operation. The main advantage offered is that of cost. The idea is to build up a bank of nDrites, one per instrument type, that are integrated and shipped with the individual instruments in question. This will then alleviate the need for expensive on-site visits and provide the desired LR-MAS connectivity. The research team already have nDrites in operation with respect to two instrument types (an auto-sampler and an ICP-MS[2]).

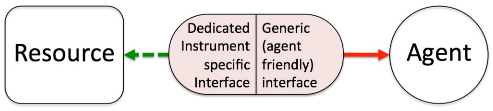
Resource
Dedicated Instrument specific Interface
Generic (agent friendly) interface
Agent

Fig. 1. nDrite smart agent enabler

The main contributions of this paper are thus: (i) the concept of nDrite smart agent enablers that facilitate multi-agent laboratory resource interconnectivity, (ii) the associated formalism that provides for the generic operation of nDrites, and (iii) two case studies illustrating the utility of the nDrite concept (the first currently in production, the second under development). The rest of this paper is organized as follows. In Section 2 some related work is presented. Our Laboratory Resource Multi-Agent System (LR-

[1] `http://www.csols.com/wordpress/`.

[2] The autosampler is manufactured by Teledyne CETAC Technologies, `http://www.cetac.com`, while the ICP-MS is manufactured by Perkin-Elmer, `http://www.perkinelmer.com/`.

MAS) framework, including the nDrite concept, is presented in Section 3. In Section 4, we detail the communication method for our LR-MAS and how nDrites handle communication aspects. The operation of the framework is then illustrated using two nDrite application case studies. The first (Section 5) is an analytical monitoring case study agent, the second (Section 6) is a resource monitoring application agent that operates using a data stream classifier. The paper concludes with some discussion in Section 7.

2 Previous Work

The notion of the pervasive, service rich and interconnected scientific laboratory has long appealed to scientists and laboratory managers of all kinds [10, 22]. Many scientific laboratory processes have traditionally involved using a number of separate, but interconnected tasks, performed by different systems and services (often bespoke) with little support for automated interoperation or holistic management at the laboratory level. To facilitate this interconnectivity, early work was directed at service oriented infrastructures using *Grid* (and later, *Cloud*) computing [8, 9, 15, 24], whereby laboratory equipment, high-performance processing arrays, data warehouses, and *in-silico* scientific modelling was wrapped, and managed, by a service-oriented client [8, 9]. The main focus was that of a "service marketplace" used to discover different services [19] and to schedule or provision their use, as well as to provide support for tasks such as: security [2], notification [17], and scheduling [24]. The need for intelligent, autonomous support for such Grid infrastructures has been well documented [8, 9, 14, 18, 24, inter alia].

The Grid Computing based laboratory infrastructure idea has now been superseded by the emergence, and wide-scale adoption, of Web Services, and consequently MAS, which exploit many of the standards used for the web, and resolved many problems of interoperability between organisations that can effect grid based approaches. This migration was essential to mitigate some of the pragmatic challenges with the interconnection of services within an Open Agent Environment [29]; however, the flexible interoperation of systems and services (developed by different stakeholders with different assumptions) is still a challenge. This motivated the adoption of a wrapper-based approach to support wide spread usability within the nDrite concept.

The laboratory instrument MAS vision thus provides for the automation of process models and workflows [28, 19]; sequences of processes that can occur both serially and in parallel to achieve a more complex task. The laboratory workflow concept has been extensively researched. The fundamental idea is that of a collection of software services, whereby each service is either a process (often semantically annotated [8, 19, 14]), or manages and controls some laboratory resource. Such workflows are typically orchestrated using editors or AI-based planning tools [19], resulting in either an instantiated workflow (one where the specific service instances are identified and used) or in an abstract workflow (one where the instantiation of the services themselves is delayed until execution time). Stein et al. [24, 25] explored the use of an agent-based approach to automatically discover possible service providers where abstract services are defined within a workflow, by using probabilistic performance information about providers to reason about service uncertainty and its impact on the overall workflow. The idea was that by coordinating their behaviours, agents could "re-plan" if the providers of other services discovered problems in their provision, such as failure, or unavailability. An interesting aspect of this workflow planning approach was the use of autonomously requesting redundant services for particularly critical or failure-prone tasks (thus increasing the probability of success). However, to facilitate the notion of autonomous control, the services themselves need to be endowed with the necessary capabilities to be self monitoring (and thus self aware), discoverable, and communicable [20].

The notion of agents supporting the management of laboratory services through interoperation and workflow (either defined a-priori or dynamically at runtime) is only possible if the agents describe and publish their capabilities, using some discovery mechanism [7]. Although many formalisms (such as UDDI, JINI, etc) have been proposed to support *white* and *yellow* page discovery systems, the discovery of agent-based capabilities based on knowledge-based formalisms describing inputs, outputs, preconditions and effects was pioneered by Sycara et. al. in the work on LARKS [26], and later with the *Profile Model* within OWL-S [1] and the machinery required to discover them [21]. However, before these descriptions and their underlying semantics can be defined, a formal model of the agent capabilities, and their properties should be modelled.

In the above previous work on the automation of process models and workflows using MAS technology, it was assumed that communication services would either be provided by some common or standardised interfaces or through some kind of mediator [27]. However, as noted in the introduction to this paper, there is no agreed communication standard currently in existence, nor is there likely to be so; whilst currently available mediators are limited to bespoke systems such as CSols' L4L system. Hence the nDrite concept as proposed in this paper.

3 The Laboratory Resource Multi-Agent System Framework

A high level view of the proposed nDrite facilitated Laboratory Resource Multi-Agent System (LR-MAS) framework is presented in Figure 2[3], where various laboratory resources are connected to nDrites, including: (i) two laboratory instruments (laser ablation systems, auto-samplers, mass spectrometers, etc.), (ii) a Laboratory Instrument Management System (LIMS) and (iii) a "links for LIMS" system (CSols' legacy mechanism for achieving instrument connectivity to LIMS, but still in operation). The figure also shows two users and a number of agents; for of which are connected directly to one or more nDrites. Two provide linkages between pairs of laboratory resources, and another two others are simply "front ends" to resources. The two remaining agents are application agents, not directly connected to nDrites: one is an Instrument Failure prediction agent and the other an Analytical Monitoring agent. We introduce Ag to denote the set of all possible agents in a LR-MAS, where $Ag = \{ag_1, ag_2, \ldots, ag_n\}$.

As noted in the introduction to this paper the interconnectivity between agents and laboratory resources in our LR-MAS is facilitated by the nDrite smart agent enablers (see Figures 1 and 2). The nDrites can be considered to be wrappers for laboratory resources in the sense that they "wrap" around a laboratory resource to make the laboratory resource universally accessible within the context of a MAS (LR-MAS). As such,

[3] This figure represents a high level vision; in practice the connectivity/operation will be more restrictive for reasons of data confidentiality and business efficacy.

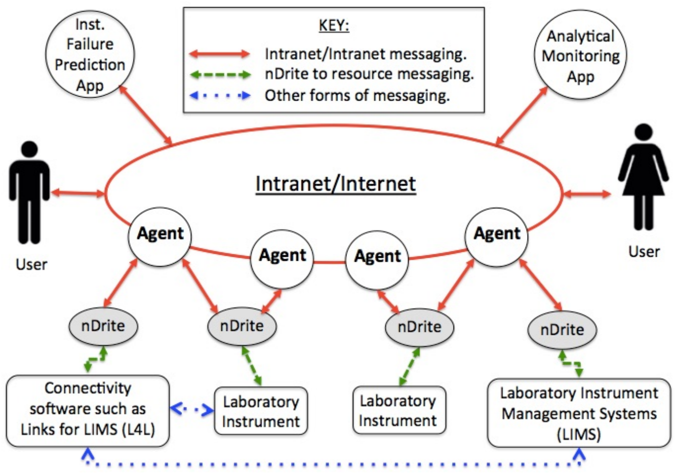
Inst. Failure Prediction App
KEY:
Intranet/Intranet messaging.
nDrite to resource messaging.
Other forms of messaging.
Analytical Monitoring App
Intranet/Internet
User
User
Agent
Agent
Agent
Agent
nDrite
nDrite
nDrite
nDrite
Connectivity software such as Links for LIMS (L4L)
Laboratory Instrument
Laboratory Instrument
Laboratory Instrument Management Systems (LIMS)

Fig. 2. nDrite facilitated Laboratory Resource Multi-Agent System (LR-MAS) configuration

nDrites can be viewed as being both the agent actuators and sensors for the laboratory resources with which they may be paired. This section provides detail of the nature of nDrites. More specifically, a formalism is presented to enable the LR-MAS vision given above. The section is organised as follows. Sub-sections 3.1 and 3.2 present the formalism with respect to laboratory resources and nDrites (in their role as actuators and sensors), respectively.

3.1 Laboratory Resources

As already noted, individual laboratories comprise a number of laboratory resources. We introduce the set of laboratory resources as $L = \{L_1, L_2, \dots, L_n\}$. Each laboratory resource has a set of one or more actions that the laboratory resource can perform. The complete set of possible actions that laboratory resources can perform is denoted by $Ac = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$. To find the set of actions an individual resource L_i can perform we use the partial action function $\texttt{LRact} : L \mapsto 2^{Ac}$. Given that there are many different types of laboratory resources (laboratory instruments, robots, data systems, and so on) resources can be grouped into a set of categories $T = \{T_1, T_2, \dots, T_n\}$, where each T_i is some subset of L ($T_j = \{L_p, L_q, \dots, L_z\}$). Each category is referred to as a *laboratory resource type*. Thus $\forall T_j \in T$, $T_j \subseteq L$ and $\forall L_i \in T_j$, $L_i \in L$. The intersection of the actions of all laboratory resources of a particular laboratory resource type are called the *critical actions* for that type, denoted $Ac^{\cap T_j}$ where for type T_j: $\bigcap_{\forall L_i \in T_j} \texttt{LRact}(L_i) = Ac^{\cap T_j}$. Note that individual resources can feature other individual actions that are not shared through the critical action set.

3.2 nDrites

The principal function of nDrites is to provide MAS connectivity without exposing the detailed operation of individual laboratory resources of many different kinds and the many different data formats. Recall that laboratory instruments are produced by many different vendors each using proprietary data formats; there are no standardised language or communication protocols for these different resources. Therefore, nDrites are used as wrappers for laboratory resources to provide a standardised method for communicating data from, and exerting control over, every *nDrite enhanced* laboratory resource. As such, nDrites can be viewed as both actuators and sensors. A formal definition of the operation of nDrites is presented below: initially in the context of nDrites as actuators and later in the context of nDrites as sensors.

nDrites as Actuators The set of nDrites are denoted as $Den = \{D_1, D_2, \ldots, D_n\}$, and the set of possible nDrite actions that the complete set of nDrites Den can expose is $Dc = \{\delta_1, \delta_2, \ldots, \delta_n\}$. The following partial nDrite *action function* defines the set of nDrite actions that a given nDrite can expose `DenAct`: $Den \mapsto 2^{Dc}$. Some nDrite actions may only be possible with respect to particular laboratory resources, others will be *critical actions* shared across a single laboratory resource type or a number of types. To find the set of laboratory resource types to which an nDrite action may be applied we use the function `pos` : $Dc \mapsto 2^T$. Actions may also be sequenced to define workflows.

Each nDrite action δ_i requires a corresponding *action object*, which details all the necessary parameters for δ_i to operate successfully. The set of nDrite action objects is $Oa = \{oa_1, oa_2, \ldots, oa_n\}$. Each nDrite action object has a class type whereby each object belongs to a class which in turn defines the nature of the object. The set of nDrite object class types is given by $Ot = \{ot_1, ot_2, \ldots, ot_n\}$. The class type of each nDrite action object is found by the following function `type`: $Oa \mapsto Ot$. To find out which class type is required for each nDrite action δ_i, we use the object requirement function `req`: $Dc \mapsto Ot$ (we assume only one object type is required for each nDrite action).

Recall that individual laboratory resources are likely to perform individual actions in different ways. Hence, at the resource end, nDrites have bespoke interfaces (see Figure 1). As such, nDrites are paired with individual laboratory resources (recall Figure 2). An nDrite D_j and a laboratory resource L_i that are connected together are thought of as an *agent enabling pair*: $AEP_k = (L_i, D_j)$. The set of all agent enabling pairs is defined as $AEP = \{AEP_1, AEP_2, \ldots, AEP_n\}$.

Consequently, given an nDrite-laboratory resource pairing, the nDrite functionality can be mapped onto the resource functionality. Additionally, note that an nDrite action δ_i for an nDrite may also include additional software only actions. A software only action is an operation performed internally to the nDrite itself with no engagement with its paired laboratory resource (for example “return the nDrite identification number”). The set of software only actions are $S = \{s_1, s_2, \ldots, s_n\}$. Therefore nDrite actions map onto zero, one or many laboratory resource actions and zero, one or many software only actions[4]. To find the set of laboratory resource and/or software only actions that occur when an nDrite action is called, we use the partial nDrite exposure function `exp`: $Aep \times Dc \mapsto 2^{Ac} \cup 2^S$. Given $L_i \in T_k$ then $\forall \delta_k$ where $T_k \notin$`pos`(δ_k), the following holds: `exp`$((L_i, D_j), \delta_k) = \emptyset$. That is, zero laboratory resource and software only actions occur when an nDrite action δ_k is attempted to be invoked on an nDrite that cannot perform it. Should an nDrite D_j want to perform an action $ac \in Ac$ on

[4] Note that the number of exposed nDrite actions can therefore be greater than the number of instrument actions.

its paired laboratory resource L_i, then it calls the function $\texttt{Perform}(ac, L_i)$. Should an nDrite D_j want to perform an action $ac \in S$ on itself, then it calls the function $\texttt{Perform}(ac, D_j)$. In both cases a Boolean is returned to indicate whether the action was successful (true) or not (false). We do not describe in detail what occurs in the `Perform` function due to the bespoke interface with the laboratory resource.

To summarise, nDrites can expose all possible actions that a laboratory resource can provide, as well as expose more software only actions. Additionally, nDrites can lower the computational burden for associated agents by exposing sequences of software and laboratory resource actions (workflows). In this manner, nDrites *enhance* the capabilities of the laboratory resources that they are attached to. Of course, for agents to trigger nDrites to perform functions, the agents must know what nDrite actions each nDrite provides. We assume this discovery capability is provided by a yellow pages agent (see [4]), which sits in the LR-MAS (not shown in Figure 2).

nDrites as Sensors For agents to work correctly with nDrites (and therefore the laboratory resources they are connected to), nDrites need to not only be actuators but also sensors. Therefore nDrites map laboratory resource actions into objects that can be understood in our LR-MAS. Previously we mentioned that nDrites, in their actuator role, receive nDrite action objects, which are required for nDrites to perform actions. Concurrently nDrites act as sensors and produce *nDrite sensor objects*. The set of nDrite sensor objects are $Os = \{os_1, os_2, \ldots, os_n\}$. Each object has a class type, where the set of object class types are defined as $Ot = \{ot_1, ot_2, \ldots, ot_n\}$ (note that this is the same definition as object types for nDrite action objects). The type of each sensor object is found by the following function $\texttt{type} : Os \mapsto Ot$. The set of sensor objects that an nDrite maps a set of laboratory resource actions onto, is found using the function $\texttt{Sen}\colon 2^{Ac} \times \mathbb{N} \mapsto 2^{Os}$, where the natural number represents the current time point.

Every nDrite D_j collects the nDrite sensor objects it generates in an associated nDrite sensor database (SDB_j)[5] that grows monotonically over time (timepoint $t = 1$ occurs when the nDrite is turned on). Depending on the end users needs, nDrite sensor databases can be local to the nDrite itself, sit on a laboratory server, or be in the cloud. The sensor database is defined as:

Definition 1: **nDrite Sensor database.** *: The database SDB_j for an agent enabling pair (L_i, D_j) holds a set of nDrite objects Osi (where $Osi \subseteq Os$), that have been generated by D_j because L_i has performed the actions LAc (where $LAc \subseteq Ac$).*

$$SDB_i^t = \begin{cases} \emptyset & \textit{iff}\ t = 0, \\ SDB_i^{t-1} \cup Sen(LAc, t) & \textit{iff}\ t > 0 \textit{ and } Sen(LAc, t) \neq \emptyset, \end{cases}$$

For nDrites to be sensors for agents, an agent needs to be able to access the objects in the nDrites database. Therefore, included in the software only actions of each nDrite are the following database access functions:

– $\texttt{GetObjectsByOccurances}_i(2^{Ag} \times 2^{Ot} \times \mathbb{N}) \mapsto 2^{Os}$. Returns the most recent n objects of the given object types that occurred in the SDB_i where $n \in \mathbb{N}$.

[5] Additionally there exists an nDrite action database for an nDrite D_j, denoted ADB_j, which holds nDrite action objects.

– `SubscribeToObjects`$_i(2^{Ag} \times 2^{Ot} \times \mathbb{N})$. Causes $ag \in Ag$ to subscribe to receiving automatic updates concerning sensor objects, saved by nDrite D_i in its database SDB_i, which are of the desired object types, until the given timepoint $n \in \mathbb{N}$.
– `UnSubscribeFromObjects`$_i(2^{Ag} \times 2^{Ot} \times \mathbb{N})$. Causes $ag \in Ag$ to unsubscribe to receiving automatic updates concerning sensor objects, saved by nDrite D_i in its database SDB_i, which have the desired object types, until the given time point $n \in \mathbb{N}$. If $n = 0$, then the agent is completely unsubscribed.

Additional functions required for the nDrites to operate successfully as agent sensors are as follows:

– `GetSubscribers`$_i(2^{Ot}) \mapsto 2^{Ag}$. Receives a set of object types and returns the set of agents that have subscribed to these object types.
– `GetNextAction`$(L \times \mathbb{N}) \mapsto Ac$. Receives a single laboratory resource and a time limit $n \in \mathbb{N}$, and returns the next laboratory action that occurs before the timelimit. If the laboratory resource performs no recognised action within the time limit then $null$ is returned.
– `Connected`$(2^{Ac} \times 2^{Ac}) \mapsto \{true, false\}$. Returns whether the first set of laboratory resource actions are connected to the second set of laboratory resource actions ($true$) or not ($false$). The two sets are connected if: (i) they form a series that can be converted into an nDrite sensor object; or (ii) they form a series that, when further nDrite sensor objects are added, can be converted into an nDrite sensor object. Also, $true$ is returned if the first set of laboratory resource actions are the empty set. $False$ is returned if the second set of laboratory resource actions are the empty set or if both sets are empty.
– `nDriteAdvertisingObjects`$_i(2^{Ot}) \rightarrow \{true, false\}$. Returns whether D_i is advertising that it can update the agents on the given set of object types ($true$) or not ($false$). Again, it is assumed that this advertisement is performed using a yellow pages agent.
– `CollectSensorObjects`$_i(\mathbb{N}) \rightarrow 2^{Os}$. Returns the objects from the database SDB_i that have occurred since the time point $n \in \mathbb{N}$.

4 LR-MAS Communication

So far we have shown that nDrites have the available functionality to be agent actuators and sensors. Note that the nDrite concept isn't simply allowing an agent to perform an action and then observe the result. nDrites allow agents to subscribe to nDrite objects, which may be generated from real world actions (e.g. a user turns an instrument off), or from other agents (e.g. another agent requests the instrument to analyse some samples). Different agents maybe interested in different actions, and so a complicated LR-MAS occurs. As nDrites are separate software entities to agents, there needs to be a communication mechanism available for the agents to utilise the actuator and sensor capabilities of the nDrites. In 4.1, we detail the message syntax between nDrites and agents. Note that the associated message syntax for agent to agent communication is considered to be out of the scope of this paper, however this can clearly be achieved using a FIPA compliant agent communication language. Sub-sections 4.2 and 4.3 show:

(i) how nDrites, in their role of agent actuators, handle incoming messages; and (ii) how nDrites, in their role as agent sensors, produce messages that get sent to agents. Finally, Sub-section 4.4 gives a brief definition for LR-MAS agents.

4.1 nDrite Message Syntax

The LR-MAS given in Figure 2 features a set of communicating entities (agent-nDrite pairs). Messages are sent between these entities, from the set of possible messages, denoted by $M = \{m_1, m_2, \ldots, m_n\}$. Each message contains *meta deta* (denoted by MD), a set of *nDrite actions and nDrite action objects pairs*[6] (NAP, where a single pair is indicated by the tuple $\langle \delta_i, oa_k \rangle$), and a set of *nDrite sensor objects* (NSO). We assume that the meta data must include two functions `Sender` and `Receiver` that returns an entity in either the set of nDrites Den or the set of agents Ag.

Definition 2: *An* **nDrite system message** *is a tuple denoted* $m_i = \langle MD, NAP, NSO \rangle$ *where the following holds:*

1. `Receiver`*(*MD*)* $\in Ag \cup Den$
2. *Sender(*MD*)* $\in Ag \cup Den$
3. *If* `Receiver`*(*MD*)* $\in Ag$ *then* `Sender`*(*MD*)* $\in Den$
4. *If* `Receiver`*(*MD*)* $\in Den$ *then* `Sender`*(*MD*)* $\in Ag$
5. *If* $NAP \neq \emptyset$ *then* $\forall \langle \delta_i, oa_k \rangle \in NAP$ *the following holds:*
   (a) $\delta_i \in Dc$; *(b)* $oa_k \in Oa$; *(c)* $oa_k \in \texttt{req}(\delta_i)$
6. $NSO \subseteq Os$

Thus an nDrite system message must have a designated receiver and sender (conditions 1 and 2). One out of the sender and receiver one must be an agent, while the other must be an nDrite (conditions 3 and 4). For each nDrite action object pair (NAP), the nDrite action called for must be valid (condition 5(a)), the paired nDrite action object must be valid (condition 5(b)) and the paired nDrite action object must be required by the nDrite action they are paired with (condition 5(c)). Finally, the nDrite sensor objects NSO that are provided must be part of the sensor object set Os (condition 6).

4.2 Sending messages to nDrites

In the *agent actuator* context, the nDrites will have to deal with many incoming messages from agents. In Algorithm 1, we present our general nDrite procedure for dealing with an incoming message. The algorithm starts with the message being unpacked (line 5). Then two sets are initialised, one for the set of nDrite actions that complete successfully (line 6) and another for the nDrite actions that do not complete successfully (line 7). As nDrites are providing wrappers for laboratory resources (instruments, LIMS, etc), an nDrite action can fail through no fault of the nDrite software. For example, a laboratory instrument message could be blocked, or the server that hosts a LIMS could fail. Therefore each nDrite records which actions have succeed and which have failed (so as to help the error recovery process for the agents within our LR-MAS).

[6] nDrite action object pairs are the objects that are saved in the nDrite action database.

Algorithm 1: The `nDriteReceive` algorithm that handles an incoming message for the nDrite D_j that is paired with the laboratory resource L_i.

```
1:  function nDriteReceive($m_i$)
2:  Input: $\langle m_i \rangle$; where $m_i$ is the received message.
3:
4:  begin;
5:  $m_i = \langle MD_i, NAP_i, \emptyset \rangle$;   // Unpack the message. No sensor objects from agents
6:  $succ = \emptyset$;                   // Set of successful actions
7:  $fail = \emptyset$;                   // Set of failed actions
8:  $p = 0$;                              // Integer count variable for nDrite actions
9:  $t = 0$;                              // Current timestamp that automatically updates
10: $complete = false$;                   // Boolean that notes whether the last action completed or not
11: $os_j \subset Os$;                    // nDrite sensor object defined
12:
13: if Receiver($MD_i$) $\neq D_j$ then
14:     return $null$;                    // If this nDrite is not the intended recipient then quit
15: end if
16: while $p < |NAP|$ do
17:     $\langle \delta, oa_k \rangle_p \in NAP$;
18:     if $\delta \in$ DenAct($D_j$) and ($oa_k =$ req($\delta$)) then
19:         $q = 0$;                      // Integer count variable for individual actions
20:         $ADB_j^t = ADB_j^{t-1} \cup \langle \delta, oa_k \rangle_p$;
21:         while $q < |$exp($(L_i, D_j), \delta$)$|$ do
22:             $ac_q \in$ exp($(L_i, D_j), \delta$);
23:             if $ac_q \in S$ then
24:                 $complete =$Perform($ac_q, D_j$);    // I.e. $ac_q$ is a software only action
25:             else
26:                 $complete =$Perform($ac_q, L_i$);    // I.e. $ac_q$ is a laboratory resource action
27:             end if
28:             if $complete = true$ then
29:                 $\langle \delta_i$, error information$\rangle \in fail$;
30:             else
31:                 $\langle \delta$, success information$\rangle \in succ$;
32:             end if
33:             $q++$;
34:         end while
35:     else
36:         $\langle \delta_i$, error information$\rangle \in fail$;
37:     end if
38:     $p++$;
39: end while
40: $fail, succ \in os_j$;       // Add the success and fail information to an nDrite sensor object
41: $m_j = \langle MD_j, \emptyset, \{os_j\} \rangle$;                 // Add sensor object to return message
42: Receiver($MD_j$) = Sender($MD_i$);
43: Sender($MD_j$) = Receiver($MD_i$);
44: Send $m_j$;
45: end;
```

The first thing an nDrite should check when a message is received, is whether it was the intended receiver (line 13 in Algorithm 1). If it was not the intended receiver the message is ignored (line 14), otherwise the message is processed (line 16 onwards). When processing the message, the nDrite takes one nDrite Action-object Pair ($\langle \delta, oa_k \rangle$) at a time (line 17). If this nDrite can perform the required nDrite action δ, and the required nDrite action object has been received (line 18), then δ is processed. Whenever an nDrite action object pair is to be processed, this is saved into the nDrite action database (line 20), so that a record of the system history is available. The nDrite processes δ by converting it into a sequence of laboratory resource and software actions via the `exp` function (line 20). If the next action ac_q is a software action, then it is performed on the nDrite (line 24), otherwise it is performed on the laboratory resource (line 26). The boolean $complete$ stores details on whether ac_p completed successfully. If any of the actions from the `exp` function are unsuccessful, then the original nDrite action δ (and information on the error) are added to the list of nDrite actions that failed (line 29), otherwise the original nDrite action δ is added to the list of nDrite actions that succeeded (line 31). This process continues until all the nDrite actions in the NAP set have been dealt with (line 16)[7]. Finally, the nDrite builds and sends a message m_j to inform the agent of what actions succeeded and what failed (lines 40 to 44).

4.3 Sending Messages to Agents

In the context of nDrites operating as sensors for agents, Algorithm 2 presents the general nDrite sensor algorithm. The algorithm takes as input the laboratory resource L_i that is pared with the nDrite D_j. Therefore the agent-enabling pair is set as (L_i, D_j). The algorithm begins by launching a database monitoring thread (line 9), the purpose of which is to monitor this nDrite's sensor database and send updates to the subscribing agents once sensor objects of the correct type appear in the database (this thread is described in more detail later). The main function then processes sequences of laboratory resource actions (describing a workflow) until termination (line 10). The Φ variable holds the current laboratory resource action series (workflow) that is being recorded[8]. This action series is initially set to empty (line 8).

When processing an action series (workflow) the first laboratory resource action is added to the current laboratory action series, as the `Connected` function always returns true when the current series is empty (line 12 and 13). Next the nDrite checks whether it advertises that it can update agents on the nDrite sensor objects that would appear from the conversion of the current action series (line 23). If so, these converted objects are added to the nDrite's sensor database (line 24), as monitored through the `nDriteMonitorDB` function. Next the nDrites waits until $timelimit$ for the next

[7] Note that if the instrument is currently busy, then the `perform` function will return false and the agent will be alerted through the error information stored in $fail$.

[8] A laboratory resource action sequence (workflow) can be processed by the nDrite as a collection; for example a sample analysis by a laboratory instrument. Single instrument actions can be: move to the next sample; send this sample for analysis; record sample results; move to next sample; etc. Some of this information maybe useful to some agents who want real time updates but other agents maybe "happy" to just have information on a collection of actions.

Algorithm 2: The `nDriteMonitor` algorithm allows the nDrite D_j to monitor the laboratory resource L_i and convert any laboratory resource actions into LR-MAS understandable nDrite sensor objects. Once converted, the nDrite will update any agents that have subscribed to these nDrite sensor object types.

```
1: function nDriteMonitor($(L_i)$)
2: Input: $\langle L_i \rangle$; where $L_i$ is the Laboratory resource to monitor.
3:
4: begin;
5: $p = 0$;                 // Integer count variable for nDrite action
6: $t = 0$;                 // Current timestamp that automatically updates
7: $timelimit$              // A predefined integer to wait for the next lab resource action
8: $\Phi = \emptyset$;      // Laboratory action series initialised
9: start nDriteMonitorDB() in new thread
10: while nDrite not terminated do
11:     $\alpha_p = \texttt{GetNextAction}(L_i, timelimit)$
12:     if Connected($\Phi, \{\alpha_p\}$) then
13:         $\Phi_p = \alpha_p$;     // Action is added to action series
14:         $p++$;
15:     else if $\alpha_p \neq null$ then
16:         $\Phi = \emptyset$;      // This action series has ended
17:         $\Phi_0 = \alpha_p$;     // A new action series is initialised with the last action
18:         $p = 1$;
19:     else
20:         $\Phi = \emptyset$;      // This action series has ended
21:         $p = 0$;
22:     end if
23:     if nDriteAdvertisingObjects(type(Sen($\Phi, t$))) then
24:         $SDB_j^t = SDB_j^{t-1} \cup$ Sen($\Phi, t$);
25:     end if
26: end while
27: end;
28:
29: function nDriteMonitorDB()
30: begin;
31: Integer $s = 0$;                       // Last timestamp checked
32: while nDrite not terminated do
33:     $\Gamma = \texttt{CollectObjects}(s)$;
34:     $s =$ current time;
35:     for each $os_i \in \Gamma$ do
36:         for each $ag_j \in \texttt{Subscribers}(\texttt{type}(os_i))$ do
37:             $m_k = \langle MD, \emptyset, \{os_i\} \rangle$;
38:             $\texttt{Sender}(MD) = D_j$; $\texttt{Receiver}(MD) = ag_j$;
39:             send $m_k$;
40:         end for
41:     end for
42: end while
43: end;
```

laboratory resource action in the series occurs (line 11). If it does not occur before $timelimit$ then the laboratory resource action will be set to $null$ (the current workflow has been completed), so `Connected` will return $false$ (line 12) and the actionSequence will be broken (line 20 and 21). Conversely if another laboratory action is found within the time limit (line 11), then if `Connected` returns true, the new action α_p is added to the sequence Φ and the process continues (lines 13 and 14). If `Connected` returns false, then α_p is not added to the current sequence, which completes (line 16), and instead, α_p becomes the first action of a new sequence (lines 17 and 18).

The `nDriteMonitorDB` thread continues to run until the nDrite terminates. The first part of the continuous loop collects nDrite sensor objects into Γ, which have occurred in this nDrite's database since the last time it checked (line 33). The last check time is then updated (line 34). For every nDrite sensor object os_i found (line 35), and for each agent ag_j that subscribes to updates concerning the objects of the type `type`(os_i) (line 36), a message is sent to each agent ag_j to inform it of the update (lines 37 to 39).

4.4 Definition of LR-MAS Agents

As discussed, there are extensive possibilities for LR-MAS agents, so we make no assumptions regrading their structure. At a highlevel, LR-MAS agents are defined as:

Definition 3: *An* **nDrite enabled LR-MAS agent** *is an autonomous software component that:*

- *Takes as input messages of the form* $\langle MD, NAP, NSO\rangle$
- *Sends messages of the form* $\langle MD, NAP, NSO\rangle$

How agents interpret nDrite sensor objects, and why they would build nDrite action objects is entirely up to them. Individual agents can perform a variety of tasks limited only by the nDrite actions implemented. The current classes of agent focused on for production are: (i) Discovery Agents, (ii) System Configuration Agents, (iii) Analytical Monitoring Agents and (iii) Instrument Monitoring Agents. We now provide two real world examples of nDrite usage (the two App agents in Figure 2). As Figure 2 shows, these two agents can be present in the same LR-MAS and connect to the same nDrites.

5 The Analytical Monitoring Case Study (Case Study 1)

Our first case study is focused on the "AutoDil agent" currently in operation (Figure 2). AutoDil uses two nDrites: (i) an Inductively Coupled Plasma Mass Spectrometer (ICP-MS) nDrite, denoted D_{icp}, and (ii) an autosampler nDrite[9], denoted D_{as}. The purpose of the AutoDil agent is to ensure any samples from the autosampler found to be "over-range" by the ICP-MS instrument are rediluted and sent for reanalysis. An ICP-MS analyses many samples, one after the other. A collection of samples is know as a run. When a run has been completed many laboratory resource actions have been performed, which are converted by the ICP-MS nDrite D_{icp} (through D_{icp}'s `nDriteMonitor` function), into a run results nDrite sensor object os_{rx} of the type ot_{rr}.

[9] An autosampler automatically feeds a liquid sample into an ICP-MS.

For the AutoDil agent ag_{ad} to do its job, it must subscribe to nDrite sensor objects of the type ot_{rr} from the ICP-MS instrument nDrite D_{icp}. Note that D_{icp} will have advertised that it can update agents with respect to objects of the type ot_{rr}, thus $\texttt{nDriteAdvertisingObjects}(\{ot_{rr}\}) = true$. When ag_{ad} receives an nDrite sensor object os_{ry} of type ot_{rr}, then it should analyse os_{ry} to see if any samples in the results run need redilution. Whenever ag_{ad} finds samples that require redilution, it:

1. Builds an nDrite action object oa_x that includes information on the dilution amounts for each sample and calls the `AddDilutions` nDrite action in D_{as} by constructing the message $m_p = \langle MD, \langle\ \texttt{AddDilutions}, oa_x\ \rangle, \emptyset\rangle$, where $\texttt{Receiver}(m_p)$ $= D_{as}$ and $\texttt{Sender}(m_p) = Ag_{ad}$.
2. Builds an nDrite action object oa_y that includes information on which samples to be reanalysed and calls the `SetupRun` nDrite action in D_{icp} by constructing the message $m_p = \langle MD, \langle\ \texttt{SetupRun}, oa_j\ \rangle, \emptyset\rangle$, where $\texttt{Receiver}(m_p)$ D_{icp} and $\texttt{Sender}(m_p) = Ag_{ad}$.

After performing both (1) and (2), the ag_{ad} waits for new objects from the ICP-MS nDrite, which may including information on samples that required further dilution.

The nDrites will deal with messages (1) and (2) through their `nDriteReceive` function. The nDrite D_{as} will convert the nDrite action `AddDilutions` through the `exp` function, to actions that its paired autosampler can understand. The purpose of these converted actions will be to tell the autosampler which samples require what level of dilution. The nDrite D_{icp} will convert the nDrite action `SetupRun`, again through the `exp` function, to actions that its paired ICP-MS instrument can understand. The purpose of these actions will be to tell the ICP-MS instrument what samples it should load from the autosampler (and therefore what data it will be collecting). The nDrites will then report to ag_{ad} what actions were successful. If all were successful then the autoDil agent knows that it should soon expect another run results nDrite sensor object os_{rz} of type ot_{rr}, which will hold information on the diluted samples.

6 The Instrument Failure Prediction Case Study (Case Study 2)

The second case study is an instrument failure prediction scenario where a dedicated agent (see Figure 2) is used to predict instrument failure using a data stream classifier trained for this purpose (as proposed in [3]). This agent is currently under development. Instrument failure within analytic laboratories can lead to costly delays and compromise complex scientific workflows [23]. Many such failures can be predicted by learning a failure prediction model using some form of data stream mining, which is concerned with the effective, real time, capture of useful information from data flows [11–13]. A common application of data stream mining is the analysis of instrument (sensor) data with respect to some target objective [5, 6]. There is little work on using data stream mining to predict the failure of the instruments (sensors) themselves other than [3] which describes a mechanism whereby data stream mining can be applied to learn a classifier with which to predict instrument failure. In our LR-MAS, an instrument failure prediction app agent implements the mechanism of [3] by communicating with other agents that are connected to nDrites (referred to as Dendrites in [3]).

7 Conclusions

We have described a mechanism to realise the benefits of MAS in the context of analytical laboratories where laboratory resources are not readily compatible with the technical requirements of MAS. Our solution is the concept of nDrites, “smart agent enablers”, that at one end feature bespoke laboratory resource connectivity while at the other end feature a generic interface usable by agents of all kinds. The vision is that of a Laboratory Resource MAS (LR-MAS). The operation of nDrites was fully described in the context of: laboratory resources, nDrites as agent actuators, nDrites as sensors, the communication mechanisms and the associated agents. The utility of nDrites was illustrated in two case studies: (i) an analytical monitoring case study for an “AutoDil agent” currently in operation; and (ii) a instrument failure prediction case study, featuring monitoring agents, that is currently under development. We believe that the proposed nDrite concept will enable the interconnected scientific laboratories of the future.

Acknowledgements

This work was conducted as part of the “Dendrites: Enabling Instrumentation Connectivity” Innovate UK funded knowledge transfer partnership project (KTP009603).

References

1. Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D. L., McDermott, D., McIlraith, S. A., Narayanan, S., Paolucci, M., Payne, T. R. and Sycara, K. *DAML-S: Web Service Description for the Semantic Web.* Proc. of ISWC, 2002.
2. Ashri, R., Payne, T. R., Luck, M., Surridge, M., Sierra, C., Aguilar, J. A. R. and Noriega, P. *Using Electronic Institutions to secure Grid environments.* 10th International Workshop on Cooperative Information Agents. p461-475, 2006.
3. Atkinson, K., Coenen, F., Goddard, P., Payne, T and Riley, L. *Data Stream Mining with Limited Validation Opportunity: Towards Instrument Failure Prediction.* 17th Int’l Conference on Big Data Analytics and Knowledge Discovery, Springer LNCS, p283-295, 2015.
4. Bellifemine, F. L., Caire, G. and Greenwood, D. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)* John Wiley & Sons, 2007.
5. Cohen, I., Goldszmidt, M., Kelly, T., Symons, J. and Chase, J.S. *Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control.* Proc 6th Symposium on Operating Systems Design and Implementation, p231-244, 2004.
6. Cohen, L., Avrahami-Bakish, G., Last, M., Kandel, A., and Kipersztok, O. *Real Time Data Mining-Based Intrusion Detection.* Information Fusion, 9(3), p344-354., 2008.
7. Decker, K., Sycara, K. and Williamson, M. *Middle-agents for the Internet.* 15th International Joint Conference on Artificial Intelligence (IJCAI’97), p578-583, 1997.
8. De Roure, D., Jennings, N.R. and Shadbolt, N. *The Semantic Grid: A Future e-Science Infrastructure.* Grid Computing-Making the Global Infrastructure a Reality, p437-470, 2003.
9. Foster, I., Jennings, N. R. and Kesselman, C. *Brain meets Brawn: why Grid and Agents need each other.* In Proc of. AAMAS, p8-15, 2004.
10. Frey, J.G., De Roure, D., schraefel, M.C., Mills, H., Fu, H., Peppe, S., Hughes, G., Smith, G. and Payne, T. R. *Context Slicing the Chemical Aether.* 1st International Workshop on Hypermedia and the Semantic Web, Nottingham, UK, 2003.

11. Gaber, M. M., Zaslavsky, A. and Krishnaswamy S. *Mining Data Streams: A Review.* ACM SIGMOD Record, 34(2), p18 - 26, 2005.
12. Gaber, M. M., Gama, J., Krishnaswamy, S., Gomes, J.B. and Stahl, F. *Data Stream Mining in Ubiquitous Environments: State-of-the-Art and Current Directions.* Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery; 4(2), p116-138, 2014.
13. Gama, J (2010). *Knowledge Discovery from Data Streams.* Chapman and Hall.
14. Gil, Y. *From data to knowledge to discoveries: Artificial intelligence and scientific workflows.* Scientific Programming 17(3): p231-246, 2009.
15. Hamdaqa, M. and Tahvildari, L. *Cloud Computing Uncovered: a Research Landscape.* Advances in Computers 86: p41-85, 2012.
16. Jacyno, M., Bullock, S., Geard, N., Payne T. R., and Luck, M. *Self-Organising Agent Communities for Autonomic Resource Management.* Adaptive Behaviour Journal. 21 (1), p3-28, 2013.
17. Lawley, R., Luck, M., Decker, K., Payne, T. R. and Moreau, L. *Automated Negotiation Between Publishers and Consumers of Grid Notifications.* Parallel Processing Letters, 13 (4). p537-548, 2003.
18. Merelli, E., Armano, G., Cannata, N., Corradini, F., d'Inverno, M., Doms, A., Lord, P., Martin, A., Milanesi, L., Moller, S., Schroeder, M., Luck, M. *Agents in Bioinformatics, Computational and Systems Biology.* Briefings in Bioinformatics, 8(1), p45-59, 2007.
19. Oinn T., Greenwood M., Addis, M., Alpdemir, M. N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M. R., Senger, M., Stevens, R., Wipat, A., and Wroe, C. *Taverna: Lessons in Creating a Workflow Environment for the Life Sciences.* Concurrency and Computation: Practice and Experience, 18(10), p1067-1100, 2006.
20. Payne, T. R. *Web Services from an Agent Perspective.* IEEE Intelligent Systems, 23(2), 2008.
21. Paolucci, M., Kawamura, T., Payne, T. R. and Sycara, K. *Semantic Matching of Web Services Capabilities.* Proceedings of the 1st International Semantic Web Conference (ISWC), 2002
22. Schraefel, M. C., Hughes, G., Mills, H., Smith, G., Payne, T. and Frey, J. *Breaking the Book: Translating the Chemistry Lab Book into a Pervasive Computing Lab Environment.* SIGCHI Conference on Human Factors in Computing Systems, April 24-29, Vienna, Austria, 2004.
23. Stein, S., Payne, T.R. and Jennings, N.R. *Flexible QoS-Based Service Selection and Provisioning in Large-Scale Grids.* UK e-Science All Hands Meeting, HPC Grids of Continental Scope, 2008.
24. Stein, S., Payne, T. R. and Jennings, N. R. *Flexible Selection of Heterogeneous and Unreliable Services in Large-Scale Grids.* Philosophical Transactions of the Royal Society A: Mathematical, Physical & Engineering Sciences, 367 (1897). p2483-2494, 2009.
25. Stein, S., Payne, T. R. and Jennings, N. R. *Robust Execution of Service Workflows using Redundancy and Advance Reservations.* IEEE Trans. Services Computing. 4(2), 2011.
26. Sycara, K., Widoff, S., Klusch, M. and Lu, J. *LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace.* AAMAS, 5(2), 173-203, 2002.
27. Szomszor, M., Payne, T. R. and Moreau, L. *Automated Syntactic Medation for Web Service Integration.* In: IEEE International Conference on Web Services, Chicago, USA, 2006.
28. Wassink, I., Rauwerda, H., Vet, P., Breit, T., and Nijholt, A. *E-BioFlow: Different Perspectives on Scientific Workflows.* Bioinformatics Research and Development: Second International Conference, Vienna, Austria, July 7-9, 2008.
29. Wens, D., Michel, F. *Agent Environments for Multi-agent Systems - A Research Roadmap.* Agent Environments for Multi-Agent Systems IV: 4th International Workshop, E4MAS 2014 - 10 Years Later, p3-21, 2014.