A Multi-Agent System for the automated handling of experimental protocols in biological laboratories

Alessandro Maccagnan*, Tullio Vardanega*, Erika Feltrin†, Giorgio Valle†, Mauro Riva‡, Nicola Cannata§
*Department of Pure and Applied Mathematics, University of Padua, via Trieste 63, 35121 Padova, Italy
†CRIBI Biotechnology Centre, University of Padua, viale G. Colombo 3, 35121 Padova, Italy
‡BMR Genomics, Via Redipuglia 21/a, 35131 Padova, Italy
§School of Sciences and Technologies, University of Camerino, Via Madonna delle Carceri 9, 62032 Camerino, Italy

***Abstract*—Software-based Laboratory Information Management Systems can handle samples, plates, instruments, users, potentially up to the automation of whole workflows. One frustrating element of this predicament is that Life Sciences laboratory protocols are normally expressed in natural languages and thus are scarcely amenable to real automation. We want to defeat this major limitation by way of a project combining Model-Driven Engineering, Workflows, Ontologies and Multiagent systems (MAS). This paper describes the latter ingredient. Our MAS has been implemented with JADE and WADE to automatically interpret and execute a structured representation of laboratory protocols expressed in XPDL+OWL. Our work has recently been tested on a real test case and will shortly be deployed in the field.**

I. Introduction

Following the explosion of automation technologies in life science experimentation, biological laboratories are drowning in data to handle, store, and analyze. High-throughput ”-omics” experiments permit to rapidly characterize whole populations of molecules (e.g. genes, transcripts, proteins, metabolites) in different samples at varying physiological/pathological/environmental conditions. Automatic processing of samples has therefore become essential in modern life sciences. For decades, Information Technology supports laboratories by means of LIMS (Laboratory Information Management Systems), software for the management of samples, plates, laboratory users and instruments and for the automation of work flow. Furthermore, formal representation of experimental knowledge is increasingly used, so that ”robot-scientists” [1] could even automatically infer new knowledge and plan subsequent experimental steps in order to confirm experimental hypotheses [2].

In natural sciences a protocol is a predefined procedural method, defined as a sequence of activities, used to design laboratory experiments and operations. In general they are still commonly described, published and exchanged in natural language and come therefore accompanied by intrinsic ambiguity, lack of structure and ”disconnection” from the surrounding execution environment. This make experiments hardly repeatable, not machine-understandable, even not easily comprehensible by a laboratory expert and most definitively not directly automatable and unfit for automated reasoning. We have therefore undertaken we have proposed to combine the unambiguous semantic of ontologies with the expressive power of workflows for formalizing protocols adopted in biological laboratories [3]. By means of workflows, protocols can be intuitively and visually represented and can be stored and shared using the XPDL standard interchange language [4]. By means of ontologies the knowledge related to the laboratory environment is directly incorporated into the workflow model and can be exchanged using the standard OWL model [5].

Building on this formal foundations (we named this meta-model COW - Combining Ontologies and Workflows) to represent laboratory knowledge and procedures we envision a Next Generation LIMS as logically composed of three main building blocks (see Fig.1): a Protocol Visual Editor, a Compiler, and a MAS Runtime Environment.

The Protocol Visual Editor allows end-users, expert of biological laboratory domain, to easily design their experiments by using controlled, well-defined domain terms to describe samples, equipments and experimental actions. End users are not required to have particular programming skills and the specifications they devise, that on the visual editor are rendered as intuitive workflows, are stored in the COW format. The editor, built on the underlying meta-model, is semantically ”cultured”, and therefore able to interpret the constructs of the meta-model. Hence, the protocols designed with the editor are syntactically and semantically correct, as the editor prevents the introduction of statements not conforming to the meta-model rules.

The end-user specification is then automatically translated

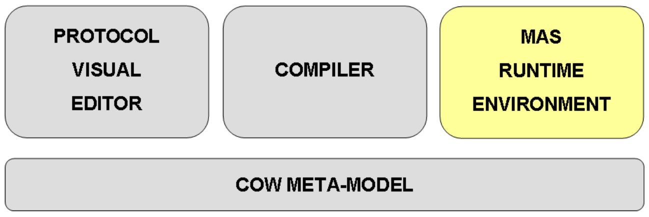

Fig. 1. The components of a Next Generation LIMS, built upon the COW meta-model

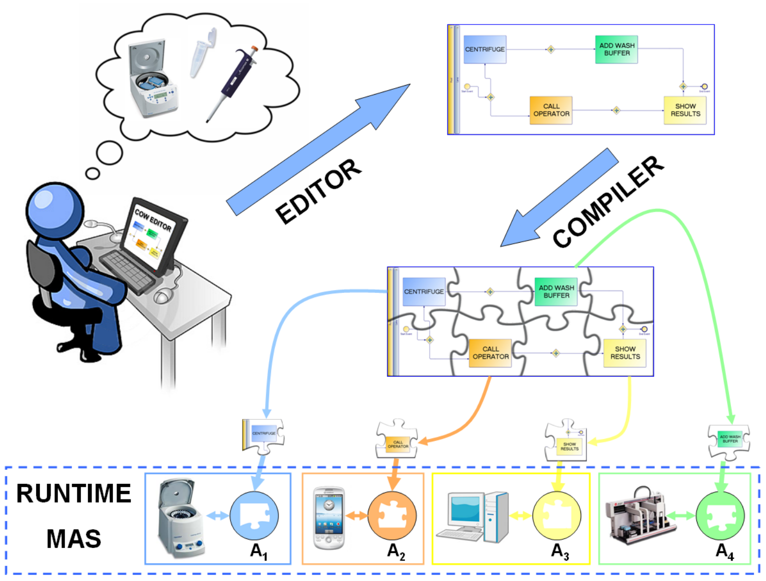

Fig. 2. The Model Driven Engineering paradigm, used in combination with software agents, for interfacing laboratory resources

by a Compiler into a runtime specification ready to be directly interpreted by a suitable execution environment. This step is analogous to the compilation process in programming languages. A program, written in a higher level programming language, perhaps RAD-designed by means of a visual editor, is translated into a lower level code that can be interpreted by a suitable runtime environment.

The MAS Runtime Environment is able to execute, constantly monitoring it, the (translated) protocol, acting as an interface and a virtualized representation of the real laboratory environment, in which the protocol will physically take place. Automated stations or human operators will be automatically invoked at runtime and they will be required to execute the actions planned in the workflows. The execution is actually delegated to a Multi Agent System, the most natural solution for a complex environment characterized by autonomous, distributed entities that need to be coordinated.

The Next Generation LIMS that we propose exploits the Model Driven Engineering paradigm [6]. The end users (see Fig.2) is able to describe the experiment model in its own language. The corresponding formal specification (in the COW meta-model) produced by the Visual Editor is then automatically translated into an executable specification that will be executed by a system of software agents. The product of the transformation is a set of Java classes that can be compiled and used directly in the runtime system. The transformation preserves the requirements and the constrainst specified by the end-users at design time.

The main requirements set on the system from a laboratory point of view are:

- traceability of the processes applied to the samples;
- automation of the laboratory processes;
- integration of heterogeneous systems and devices used in the laboratory.

Our project aims at simplifying the work of laboratory operators. With the drastic increment of formalization and automation, the room for man-made errors will be greatly reduced and all the bookkeeping activities will not absorb any more the staff time.

In this paper we introduce the Executable model of a protocol that can be interpreted by a MAS and then we concentrate our attention on the MAS Runtime Environment itself, describing its architecture, its components and its coordination. The paper is organized as follows: Section II presents the Executable model of a protocol; Section III describes the architecture of the Multi Agent System Runtime Environment supporting the execution of the protocols; Section IV presents a demo case study; Section V concludes with final remarks and possible future improvements.

II. Executable Model of a protocol

In this section we describe the Executable model of a protocol, that we have defined for coding executable protocols in our Next Generation LIMS. An essential issue in LIMS is the necessity of monitoring a protocol during its execution, saving both the information on data and on procedures. It must be underlined that the executable protocol is not the one defined by the end-user by mean of the Visual Editor and stored in the COW format, but its translation generated by the Compiler.

A laboratory protocol can be seen as the composition of precisely defined activities [7]. The executors of the activities could be instruments (e.g. a centrifuge) or laboratory staff. As an example of the huge quantity of data produced by an activity we could cite the mass of raw data generated by a single DNA sequencing experiment, that is nowadays in the order of the Terabytes. Beside the data, all the procedural steps must be tracked. Returning to the DNA sequencing

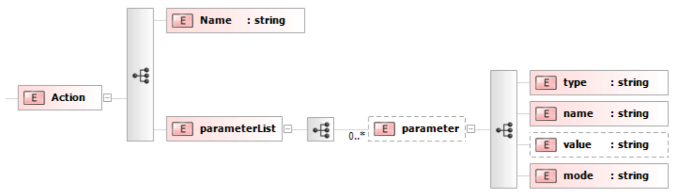

Fig. 3. XML schema for an Action

laboratory example, the protocol would probably require to execute also ”virtual” operations like converting the raw data into DNA sequences and subsequently assemble them by means of alignment algorithms.

In the typical protocol we can found three kind of activities, depending on their performer:

- those performed by a physical device, like a liquid handler workstation (e.g. Biomek FX);
- those performed by a virtual device, like an assembling software;
- those performed by a human operator, like shaking a plate or taking a sample of DNA through a swab.

Starting from these considerations we have defined a general notion of activity, called Action. An Action is defined by a name and a list of parameters. Each parameter is characterized by a name, a type and by the mode in which it is passed (read-only, read/write, write-only) (Fig.3). An Action is an atomic step in our execution model and could be seen as the simplest instruction that our MAS is able to interpret and execute

Referring again to the programming language metaphor we can assimilate an Action in the Executable model of a protocol to a single machine instruction in a machine code program. A single machine instruction can be directly executed by the processor. The definition of Action is general enough to include the three cases above mentioned. Given the heterogeneity and the complexity of the laboratory environment, this solution represents a good trade-off between the need of describing a protocol with enough granularity and the need of having a common interface for every activity involved.

Fig.4 shows the XML document describing a ”centrifugate” Action. A centrifugate Action is defined by three parameters. The parameter named ”performed”, is of boolean type and its mode is OUT, so that it is actually an output parameter of the action, representing whether the action has been successfully performed. The second and the third parameters are inputs of integer type representing respectively the g-force to be applied in the centrifugation and the centrifugation time.

It must be recalled that an Executable Action has a semantic counterpart in the Action concept, formally defined in the laboratory domain ontology of the COW meta-model supporting the Next Generation LIMS [3].

In the Executable model, a Protocol is an articulated flow of Actions. A Valid Protocol is a protocol that our runtime environment is able to interpret and to execute. A Protocol in the model could be composed using different Actions available in the runtime environment or loaded from external

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<Action>
    <Name>centrifugate></Name>
    <ontoTag>tagCentrifuge></ontoTag>
    <parameterList>
        <parameter>
            <type>boolean></type>
            <name>performed></name>
            <mode>OUT</mode>
        </parameter>
        <parameter>
            <type>int</type>
            <name>forceApplied</name>
            <mode>IN</mode>
        </parameter>
        <parameter>
            <type>int</type>
            <name>centrifugationTime</name>
            <mode>IN</mode>
        </parameter>
    </parameterList>
</Action>
```

Fig. 4. XML document for a centrifugate action

libraries. We use the XPDL [4] meta-model to describe the execution and the orchestration of different actions. In XPDL a process is a structured composition of piece of works, called Activities, that could be of various type [8]. In our system a Valid Protocol is a XPDL compliant model with some minor limitations and differences.

In order to guarantee the correct interpretation of a COW protocol we do require that every piece of work must be described by means of an Action concept. In this manner the COW meta-model is semantically enriched and we want to preserve at runtime the ontological constraints defined at design-time. In XPDL the notion of ”piece of work” is described by the concept of Activity. Hence we impose that every Activity is allowed to invoke only Actions. In order to satisfy this condition in the Executable model, we imposed two restrictions to the XPDL meta-model.

The first one is to limit the types of Activity only to Route and SubFlow. The Route activity performs no work and simply supports routing decisions among the incoming transitions and/or among the outgoing transitions. The SubFlow activity enables the reuse of processes and could be usefully used to encapsulate parts of protocols in self-contained modules.

Second, we provide a specific SubFlow (ExecuteActionW) around an invocation of an Action. The ExecuteActionW SubFlow (Fig.5) simply invokes the execution of the Action and checks if it is performed with or without errors. Actions can be executed only encapsulated within such construct.

Using only Route and SubFlow activities and using the ExecuteActionW SubFlow we can therefore ensure that every piece of work is backed by an Action concept. In the next section we will describe how we have built a MAS runtime system able to execute a Valid Protocol.

III. MAS Runtime Environment

The enactment of workflows using multiagent systems has already been proposed in the literature [9], [10], including

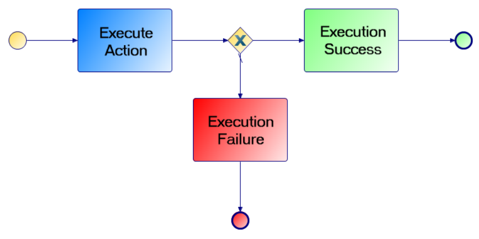

Fig. 5. The ExecuteActionW SubFlow that encapsulate the execution of actions

the development [11], [12] of workflow management systems based on the popular open source MAS platform JADE [13].

Recently, a new software platform has been proposed as an extension to JADE by the JADE development group itself. WADE (Workflow and Agent Development Environment) has been developed on top of JADE, with the implementation of new features for supporting the use of workflows in the deployment of multi-agent applications [14]. WADE includes a micro-engine embedded in a set of dedicated agents which are specifically developed for the execution of workflows defined in an extended version of XPDL. Doing this, the new engine permits to directly execute the Java code associated to a specific workflow activity. Moreover, this new tool allows to choose and assign secondary agents for the execution of subflows. Additional components have been also defined in WADE in order to manage administration and fault tolerance issues.

The main challenge in WADE consists in bringing the workflow approach from the business process level to the level of system internal logics [15]. In other words, the objective is not to support an orchestration of services provided by different systems at high level, but to implement the internal behavior of the single systems.

The new functionality offered by WADE is of special importance for us. The capabilities to run a slightly different model of XPDL workflows fits with our requirements. We hence decided to build our MAS on top of the JADE/WADE framework.

The architecture of the MAS Runtime Environment is designed to closely resemble the laboratory environment, with the additional capability of being able to interpret and execute Actions as described in Section II. The Executable Model of a protocol involves one main kind of entities. These entities are heterogeneous and distributed resources that actually expose and, on request, performs Actions. We therefore dedicated one class of agent to these entities, the Device Agent (DA). Another distinctive characteristic of the Runtime Environment is an entity that does read an executable protocol and handles its execution. A Protocol Manager agent (PM) is appointed to control this aspect. A user interface agent (APE) is designed for loading new protocols in the MAS. A Reporter Agent (RA) is built specifically as a User Interface for mobile devices.

- **DA**: controls a resource (physical or virtual)
- **PM**: executes a protocol in the MAS
- **APE**: allows the loading of new protocols
- **RA**: user interface for mobile devices

The RA agent is created at the boot of the system. For each resource in the laboratory environment that should be automatically managed from the LIMS, it is then created a DA Agent counterpart. One APE is also created in the boot phase, however two (or more) instances can co-exist without any problem. The same apply for the RA. A PM agent instead is created dynamically on user demand, being responsible for the execution of a particular protocol. Once completed the protocol, the PM agents automatically disappear from the system.

A. *Device Agent*

In the past years the literature recognized the need to explicitly embody the notion of resource in a MAS [16], [17]. A well known approach is to use the notion of "artifact". Artifacts can be considered as complementary abstractions to agents populating a MAS. While agents are goal-oriented pro-active entities, artifacts are a general abstraction to model function-oriented passive entities. MAS designers employs artifacts to encapsulate some kind of functionality, by representing (or wrapping) existing resources or instruments mediating agent activities [18]. The purpose is to encapsulate functionalities and services in suitable first class abstractions at the agent level [16]. Artifacts could be used for wrapping existing resources and therefore are a suitable model for our purpose. Particularly fitting is the Agent and Artifact model [16], in which an Artifact is structured as a set of operations.

We therefore propose a combination of a *Driver* and an Agent in order to make available in the MAS a service that can be executed by a physical or virtual resource. A *Driver* in our model actually performs the communication with a legacy resource as a centrifuge or a robotized station. Since our model is inspired from the A&A model, our Driver is structured in terms of Actions. The similarity with the model lies on the fact that a Driver represents a resource in the MAS environment (Artifact in the A&A model). The main difference is that we strictly bind an instance of a Driver with exactly one instance of a DA. As a consequence every request of Actions must be posed to a specific DA that acts as a proxy to the driver and therefore to the resource.

A Resource exposes a set of Actions, one for every functionality. In the Centrifuge example the Centrifuge is the resource itself, and it is described by means of the functionalities it exposes and hence by a set of different Actions i.e. the actions it can actually perform (e.g. "centrifugate" or "open lid"). A DA is responsible for executing the single actions, therefore it needs to knows how to physically communicate with the related resource. It needs also to communicate with other agents in order to satisfy any request for its functionalities. We therefore structured a DA in two layers as depicted in Fig.6.

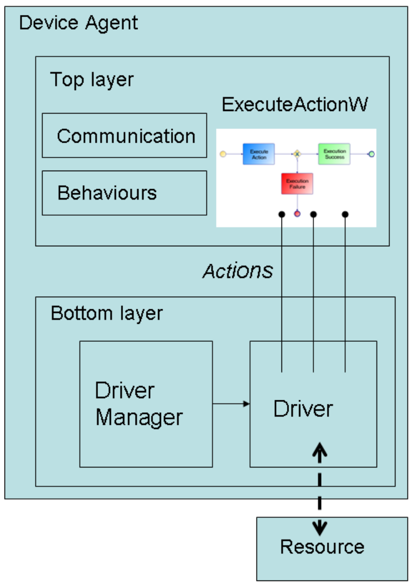

Fig. 6. Layers of the Device Agent

The bottom layer is responsible the communication with the resource using a specific driver. The top layer possesses the normal agents duties as behaviors and social capabilities.

1) Bottom layer: The bottom layer is to load, extract the related metadata, and it uses a resource specific driver. In the development of such a complex and heterogeneous system like a biological laboratory, the design of a new driver can become a hard bottleneck. Hence we made some effort to simplify the process of driver creation. In our approach, a driver could be any piece of Java code. This choice enables the reuse of legacy code as well as direct interfacing with the instrument. The only added requirement for a developer is to declare which services the driver does expose. This is done via the Java annotation mechanism, which allows to add metadata to the code.

We provide a set of annotations like @Action and @Par. Every method that is going be exposed as a first class entity in the system (Action) must be annotated with the @Action tag. In case of parameters the @Par tag should be used. As illustrated in Fig.7 the method *centrifugate* is promoted to an Action entity in the MAS. Two parameters are declared plus an extra one for the return value of the method. The XML document of Fig.4 it is actually created from the annotated *centrifugate* Java method of Fig.7.

Using a driver manager the Device Agent is able to load and extract the metadata for a driver that fulfill these requirements. During the initialization phase the agent loads the driver, analyzes the metadata and fills a set of Action objects compliant to our model. The set of these objects provides the descriptions of the capabilities of the Device agent. The last step of the initialization phase is to register itself (with the exposed capabilities) in the MAS.

```
@Action(ontoTag = "tagCentrifuge", returnName = "performed")
public boolean centrifugate(
    @Par(name = "gforce", mode = Mode.IN) int rpm,
    @Par(name = "sec", mode = Mode.IN) int sec)
    {
    //comunication with centrifuge
    }
```

Fig. 7. Example of a method annotated with an @Action tag

2) Top Layer: This layer is responsible for the interaction with other agents in the MAS, responding to to request of Actions. The main capability consists in being able to execute the ExecuteActionW SubFlow (see Fig.5). A different agent, intending to execute an action available on the interfaced device, should first retrieve the corresponding Action object. Then, this agent should ask to the DA to perform the Execute-ActionW SubFlow using as parameter the Action object and the actual parameters (if any) of the action. The DA then tries to execute the action, communicating with the resources by means of the driver. If some error occurs the caller is notified. In case of no errors, the resulting output parameters are filled in the Action object and the caller is notified.

B. Protocol Manager agent

The Protocol Manager agent is responsible for the correct execution of a protocol. It incorporates the capabilities to execute a restricted (according to Section II) XPDL protocol. Since the restriction imposed to XPDL in our Executable Model are minor, a normal WADE agent could be used. In order to develop a protocol directly in the MAS system it is therefore possible to use the WOLF tool [19]. However, in the future, we intend to translate a protocol, structured in the COW meta-model, directly in the underlying Java code.

On the launch of a new protocol a new PM is created. The first step performed by a PM is to check if the protocol can run on the current environment. Therefore the PM tries to verify the existence of every Action used in the protocol before actually starting the execution. Only if that control is successful then the execution of the protocol can take place. When the PM agent encounters an Action invocation it first finds which DA is actually able to perform it. The search is performed using the classic yellow pages system of Jade. Then, the agent delegates the execution of the ExecuteActionW SubFlow to the proper DA. The standard WADE mechanism used to enact distribuited workflow execution is applied.

If multiple protocols require the same action, the requests are queued and acted upon by the DA. The requests are then executed on FIFO bases (first in first out). In the future, using a separate scheduler more complex policies could be applied.

C. Agent Protocol Environment

The APE agent provides a user interface [UI] to laboratory operators in order to load new protocols. A protocol is enclosed in a package that contain three different categories of elements:

- main protocols as well as the sub-protocols used in them;
- local resources like images or spreadsheet files required by the activities of the protocols;

- specific external libraries used to obtain some extra feature like PDF documents generation.

APE loads a package and does visualize the content to the operator. It then extracts all the resources and does deploy them into the runtime system. It is also responsible for creating a new PM agent and to charge it with the execution of the loaded protocol.

D. *Reporter Agent*

A Reporter Agent has been built specifically to handle requests from mobile devices that provide GUIs to laboratory operators. We currently support ANDROID [20] based mobile devices using the peer-to-peer approach proposed by [21].

The RA is able to query the system and to provide information about the state of a sample processed in the laboratory. It interacts with the other agents of the runtime system and queries the database in order to determine detailed information like:

- the customer order that activate the laboratory analysis;
- the type of the biological analysis in which the sample is involved;
- the current phase of processing reached by the sample;
- the relationship with other samples produced in the laboratory for the same customer order.

Finally, after collecting all pieces of information, the RA is able also to produce a report and to send it to the operator's GUI on its mobile device.

IV. A DEMO CASE STUDY

The Paternity Test aims at establishing if a man is the biological father of an individual. A customer, willing to perform the test, places an order through a website. Afterward, DNA samples belonging to the individual and to the supposed child are collected, usually by means of buccal swabs, and sent to the analysis laboratory. When the material reaches the laboratory, some biological analysis can actually be executed, according to the protocol described in Fig.8. Through several sub-protocols, the samples are processed and, at every step, transformed into specific types of succeeding samples. In the final steps of the protocol, by DNA sequencing techniques, some data results are obtained. The DNA sequencing output is then used to compute the profile of the individuals involved in the specific test and finally a medical report that explain the results is produced by an expert.

Each action of the protocol is currently activated manually by a laboratory operator, following the workflow. In different phases of the process the operator is bounded to fill some digital resources and execute some bioinformatics analysis.

In order to test the potential of our system the protocol described above has been formalized in the COW meta-model. The graphical representation of the workflow of Fig.8 is rendered using the WOLF tool [19] of the WADE framework. Every depicted activity block could represent either a normal SubFlow or an Action invocation by means of the ExecuteActionW SubFlow.

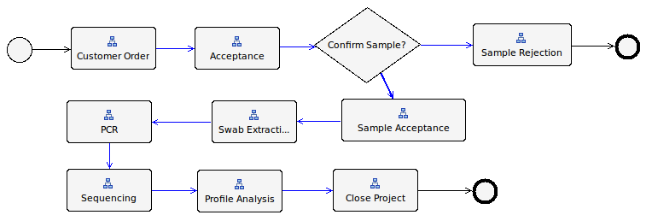

Fig. 8. The protocol formally describing the Paternity Test

Fig.9 shows a subprotocol that describes the steps involved in the PCR SubFlow. In it we can see a use of the centrifugate action of Fig.4. In the PCRCycle Action the DNA material is amplified by means of the Polymerase Chain Reaction so that its quantity becomes sufficient for the following steps of the analysis. It can be noticed that the protocols does include not only the physical processing of samples but also the management of the produced data and of the history of the sample (e.g. by mean of the DBReg Action, that interacts with a database). Doing so permits to support existing legacy systems without changing their structure.

It is worth underlining that since the PCR sub-protocol is self-contained in the SubFlow is possible to reuse it in other contexts without writing a single line of code. This drastically reduces the time needed for the implementation of new protocols.

Using the proposed approach an explicit knowledge of the concepts involved protocol exists in the system. The MAS is therefore able to interpret this knowledge and to act correctly depending on the real environment. In the case study of the paternity test only the tracking activities have been totally automated. The operator is therefore notified when he can start the physical steps, to be executed from a device. Nevertheless, with propers drivers and proper hardware, also physical actions could be automated. The system notifies with the next steps to be performed. In the example the operator is notified to perform a PCR on some specific samples. After the sequencing phase an automatically analysis is performed and the results are delivered to the laboratory operator.

In our test case a total of 31 activities are included to define the paternity protocol (included the sub-protocol *SwabExtraction, PCR, Sequencing and Analysis*). Using our model we automatized 12 of those activities. Once automatized these activities become transparent to the end user and they could be also easily reused in other protocols with minimal effort.

On respect of the initial requirement of traceability, automation and integration our test case show promising results. The requirement of traceability is easily guaranteed, and all the related - and heavy - duties are now transparent to the end user.

The second requirement of automation is met. In our test case only some activities have been automatized. The bottleneck is the legacy environment and the development of the drivers. However, also without producing drivers for the specific hardware, we automated 12 of the 31 initial activities

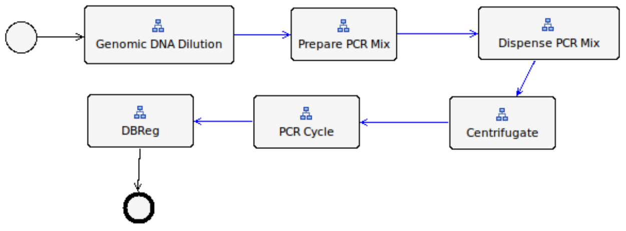

Fig. 9. The PCR subprotocol of the paternity protocol

(38%).

The last requirement is met under the constraint to produce specific drivers for the specific devices used in the laboratory.

V. Conclusion

In our vision a Next Generation LIMS will ease the management of a biological laboratory. The laboratory knowledge, including the procedural one, would be formalized and would become easy, for automatic systems, to exchange, control, analyze and exploit. On the other hand, laboratory protocols would become easy to specify, read and maintain also by domain experts.

Protocols could be statically validated in a given execution environment (i.e. "MAS interfaced" laboratory) according to specifications to be met. In this way it will be possible to know if the environment is able to execute the protocol ensuring the expected quality of service respecting all the critical constraints.

A Scheduler agent could also perform a dynamic validation and optimization of a protocol at runtime. In this way it will be possible to effectively and efficiently answer to possible sudden changes in the environment, like e.g. the breakdown of a resource. The actions to be taken in order to recover from abnormal situations could be based on a set of rules. The agents should perform some reasoning upon that rules in order to react in a proper and fast way to the changes of the environment.

It could also dynamically schedule groups of samples acting on the current state of the resources, in order to optimize the resource consumption.

The system will permit an easier tracking of all the transformations applied to the analyzed samples, including the virtual (i.e. digital) ones. Automation will be naturally enhanced: physical devices will undergo a direct control performed by software agents, in turn controlled by protocol "instructions". The integration of heterogeneous systems and instruments, including the communications with laboratory operators, also via mobile platforms, will thus become possible.

References

[1] R. King, K. Whelan, F. Jones, P. Reiser, C. Bryant, S. Muggleton, D. Kell, and S. Oliver, "Functional genomic hypothesis generation and experimentation by a robot scientist," *Nature*, vol. 427, pp. 247–252, 2004.

[2] R. D. King, J. Rowland, S. G. Oliver, M. Young, W. Aubrey, E. Byrne, M. Liakata, M. Markham, P. Pir, L. N. Soldatova, A. Sparkes, K. E. Whelan, and A. Clare, "The Automation of Science," *Science*, vol. 324, no. 5923, pp. 85–89, 2009. [Online]. Available: http://www.sciencemag.org/cgi/content/abstract/324/5923/85

[3] A. Maccagnan, M. Riva, E. Feltrin, B. Simionati, T. Vardanega, G. Valle, and N. Cannata, "Combining ontologies and workflows to design formal protocols for biological laboratories," *Automated Experimentation*, vol. 2, no. 1, p. 3, 2010. [Online]. Available: http://www.aejournal.net/content/2/1/3

[4] Xml process definition language. [Online]. Available: http://www.wfmc.org/xpdl.html

[5] Ontology web language. [Online]. Available: http://www.w3.org/TR/owl-ref/

[6] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," *FOSE '07: 2007 Future of Software Engineering*, pp. 37–54, 2007.

[7] M. Courtot, W. Bug, F. Gibson, A. Lister, J. Malone, D. Schober, R. Brinkman, and A. Ruttenberg, "The owl of biomedical investigations," in *Proceedings of the Fifth OWLED Workshop on OWL: Experiences, 2008*, xx 2008.

[8] R. Shapiro and M. Marin, *Workflow Management Coalition Workflow StandardProcess Definition Interface– XML Process Definition Language*, The Workflow Management Coalition, 99 Derby Street, Suite 200 Hingham, MA 02043 USA, October 2008.

[9] P. A. Buhler and J. M. Vidal, "Towards adaptive workflow enactment using multiagent systems," *Inf. Technol. and Management*, vol. 6, no. 1, pp. 61–87, 2005.

[10] E. Bartocci, F. Corradini, and E. Merelli, "Enacting proactive workflows engine in e-science," in *International Conference on Computational Science (3)*, 2006, pp. 1012–1015.

[11] C. V. Trappey, A. J. Trappey, C.-J. Huang, and C. Ku, "The design of a jade-based autonomous workflow management system for collaborative soc design," *Expert Systems with Applications*, vol. 36, no. 2, Part 2, pp. 2659 – 2669, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6V03-4RV7Y9W-2/2/f933cced0e8af5448692818153bb1648

[12] G. Fortino, A. Garro, and W. Russo, "Distributed workflow enactment: an agent-based framework," in *Atti del 7 Workshop dagli Oggetti agli Agenti (WOA) Sistemi GRID, Peer-to-peer e Self-*, Catania (Italia).*

[13] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. Wiley, April 2007.

[14] G. Caire, D. Gotta, and M. Banzi, "Wade: a software platform to develop mission critical applications exploiting agents and workflows," in *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 29–36.

[15] A. Poggi and P. Turci, "An agent-based bridge between business process and business rules," in *Decimo Workshop Nazionale Dagli Oggetti agli Agenti*, 2009.

[16] A. Ricci, M. Viroli, and A. Omicini, "The a&a programming model and technology for developing agent environments in mas," in *ProMAS'07: Proceedings of the 5th international conference on Programming multi-agent systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 89–106.

[17] A. Omicini, A. Ricci, and M. Viroli, "Artifacts in the a&a meta-model for multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 3, pp. 432–456, 2008.

[18] R. Kitio, O. Boissier, J. F. Hbner, and R. Ricci, "Organisational artifacts and agents for open multi-agent organisations: giving the power back to the agents."

[19] G. Caire, M. Porta, E. Quarantotto, and G. Sacchi, "Wolf - an eclipse plug-in for wade," in *WETICE '08: Proceedings of the 2008 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 26–32.

[20] Android platform. [Online]. Available: http://code.google.com/android

[21] M. Ughetti, T. Trucco, and D. Gotta, "Development of agent-based, peer-to-peer mobile applications on android with jade," *Mobile Ubiquitous Computing, Systems, Services and Technologies, International Conference on*, vol. 0, pp. 287–294, 2008.