Engineering Pluripotent Information Systems

Ivan J. Jureta
PReCISE
University of Namur
iju@info.fundp.ac.be

Stéphane Faulkner
PReCISE
University of Namur
stephane.faulkner@fundp.ac.be

Jean Vanderdonckt
BCHI Laboratory
University of Louvain
vanderdonckt@isys.ucl.ac.be

Abstract— A pluripotent information system is an open and distributed information system that (i) automatically adapts at runtime to changing operating conditions, and (ii) satisfies both the requirements anticipated at development time, and those unanticipated before but relevant at runtime. Engineering pluripotency into an information system therefore responds to two recurring critical issues: (i) the need for adaptability given the uncertainty in a system's operating environment, and (ii) the difficulty to fully anticipate and account for all possible stakeholders' requirements at development time and respond to the change of requirements at runtime. We draw on our group's research efforts over the last two years to show and discuss how pluripotency can be engineered into information systems.

I. INTRODUCTION

The term "pluripotency" in biology describes the potential of a cell to develop into different cell types depending on the environment. In line with this use we carry over the term to information systems engineering to designate a class of information systems (IS) that have two distinguishing traits:

1) *Adaptability to the variability of modules.* A pluripotent IS satisfies stakeholders' requests by selecting and coordinating the execution of self-contained modular applications (be they services as in service-oriented computing, or agents as in agent-oriented software engineering). The modular applications, i.e., modules, are selected from a pool which is not stable throughout system runtime: modules can be made available and become unavailable, modules' providers can upgrade or replace them entirely, and new providers can make their modules available at system runtime. The engineer of the pluripotent IS therefore does not know at development time the characteristics of all modules that may be involved in the IS at runtime. She engineers the IS so that it adapts to the varying availability and unanticipated characteristics of the modules.

2) *Adaptability to the variability of requirements.* Just as a pluripotent cell can develop into different cell types, and is thereby capable of serving different purposes within a living organism, a pluripotent IS can select and coordinate various sets of modules to satisfy different stakeholders' requirements. This results from the system's ability to operate with various modules (each performing some delimited function) and to combine them in various ways depending on the specifics of stakeholders' requests at runtime. Rarely, if never is the pluripotent IS made with one specific application area

in mind. Instead, pluripotent IS are particularly relevant when wide ranging sets of requirements need to be satisfied, when the engineers are unsure of what precise requirments the stakeholders will have throughout the system lifecycle, and when the engineers themselves are not developing the various modules but rely on external providers.

The infrastructure for developing and deploying pluripotent systems is available, though there is at present no fully functioning or commercially available pluripotent IS. This is due to two issues, which we revisit further below: (i) lack of agreement on standards which would ensure interoperability between modules; (ii) definition and testing of the mechanisms which enable the system to select the best possible sets of modules to satisfy to the most desirable extent stakeholders' requirements. Over the last two years our efforts focused on identifying and proposing solutions to the second issue. We have already presented various results in a fragmented manner and to different communities [1], [2], [3], [4], [5], [6], [7]. The purpose of this paper is to overview available results, place them within a single framework, outline open issues, and motivate further research. It is directed at the information systems engineering community for we see it as an appropriate setting in which to discuss and evaluate these efforts in their full scope.

a) Organization.: We start by outlining the main motives and trends in research and industry that led us to work on the engineering of pluripotent IS (§II). We then define the problem that needs to be resolved if pluripotent IS are to be relevant in industry (§III). We review our solutions to parts of the problem (§IV and §V). We close the paper by discussing the lessons learned over the last two years, summarize the conclusions, and point to directions for future effort (§VI).

II. MOTIVATION

It is well known that the engineering and management of increasingly complex IS is a key challenge in computing (e.g., [8], [9]). Relevant responses in research and industry involve a move towards systems' increased modularity, openness, distribution, and interoperability. Among the various, often overlapping approaches to building such systems, service-orientation and agent-orientation stand out in terms of accumulated experience and advances in research, availability of standards for describing and enabling the interaction between modules (be they services or agents), and uptake in industry.

Any pluripotent IS relies on a modular, open, and distributed architecture, be it service- or agent-oriented. Pluripotent IS engineering relies on available results on enabling modularity, openness, distribution and interoperability, and is not *per se* concerned on advancing these matters.

Overall, a pluripotent IS involves three basic constituents, namely, requests, composers, and modules. The system operates as follows.[1] In a pluripotent IS a *request* represents stakeholders' requirements in an unambiguous and machine-understandable format. Requests are created at runtime: stakeholders express requirements, which are then translated into requests. Each request is submitted to an automated *composer* in the IS, which then selects the individual modules whose coordinated execution fulfills best the requirements represented by the request. In selecting among different and/or competing modules[2], the composer matches stakeholders' requirements to systems' capabilities. The engineering of the composer is in such a context difficult for two reasons discussed in turn below.

A. Variability of Modules

We have noted earlier that openness is a characteristic of any pluripotent IS. By openness, we mean that some module provider that has not registered its modules with the pluripotent IS during its development can still make these modules avalable 'in' the pluripotent IS. Modules whose characteristics are unknown at development time of the pluripotent IS can therefore become involved in the pluripotent IS and be considered for selection by composers. In other words, the pool of potential modules varies and individual modules' actual properties (i.e., what function it performs, and how well it performs it) are unknown to the composers before the said modules become available at runtime. Having a composer that can operate with initially unknown modules is of evident interest in terms of maintaining and evolving an IS. To make a composer receptive to new modules, ensuring interoperability and choosing appropriate behavior for the composer are both critical. If interoperability is ensured, it will be possible for the composer to detect what function a module offers and how well it performs the function. Ensuring interoperability remained mostly outside the scope of our efforts—the reader is referred to literature on the Semantic Web, which is directly relevant for pluripotent IS whose modules are web services. We have mostly been concerned with defining appropriate behavioral mechanisms for composers, while simplifying by assuming interoperability. In other words, the issue we have been and remain interested in is what (class of) algorithms can be used to enable composers *to select modules whose coordinated execution satisfies to the most desirable extent stakeholders' requirements laid out in a request, given that at any time a different pool of modules is available*. We

present the salient characteristics of one such algorithm in the remainder of the paper.

B. Variability of Requirements

Requirements engineering consists of describing the stimuli that the future system may encounter in its operating environment and defining the system's responses according to the stakeholders' requirements. The more potential stimuli are anticipated and accounted for during development, the less likely a discrepancy between the expected and the observed behavior and quality of the system. Hence a longstanding concern in RE on requirements completeness (e.g., [10], [11]): i.e., ensuring that the requirements specification accounts for all the relevant stakeholder requirements and environment properties and behaviors. Perfect completeness is rarely claimed in practice. Stakeholders learn as they participate in the system development project and revise their requirements, which they continue doing while using the deployed system and thereby revise expectations depending on the interactions with the system. Revised requirements may then lead to the reengineering of the system so as to accommodate new requirements. Requirements are often well known for IS focused on very specific tasks and used in industry, and their variability is manageable through periodic change of the relevant systems. Requirements tend to be considerably less clear and stable in consumer-oriented IS, such as, e.g., photo and video sharing applications, and office productivity suites. They are usually anticipated and/or identified in a reverse manner (i.e., technology allows new functionality, so that the functionality is provided independently of being requested by the users). That individuals' goals and preferences are variable has long been recognized in particular in research on decision making and marketing. Once one acknowledges that requirements are likely to vary instead of remaining stable, it becomes apparent that the ideal approach is to engineer a system that places very few bounds on the content of requirements that stakeholders can submit to it. Given modularity, openness, and interoperability, it becomes relevant to enable an IS during development to accommodate various functionality through modules and requests for functionality from stakeholders. Engineering pluripotency into an IS allows the engineer to ensure that the system responds to one main requirement, which is *to be able to respond to various requirements of the stakeholders* instead of limiting the system to a restricted set of requirements identified at development time. Pluripotent IS are thus an alternative to available solutions to requirements variability, with the advantage of being conceptualized to primarily address requirements variability. The very notion of pluripotency, that is, the possibility to develop in various ways so as to satisfy various purposes, therefore fits well the present discussion.

C. Industrial Relevance of Pluripotent IS

Benefits of increased modularity, openness, distribution, and interoperability are recognized in industry, among other through adoption of service-oriented architectures for IS [12].

---

[1]Details of the architecture that can enable the operation of pluripotent IS are given elsewhere, both for service-oriented computing [3] and agent-oriented computing [2].

[2]Different modules perform different functions. Competing modules all perform the same function, though at different quality levels.

Experiences with service implementations (e.g., [13], [14]) indicate that service-orientation does facilitate application integration. Experiences in the German banking sector show that facilitated integration opens new possibilities in how banking is done [13]:

> "To decrease costs and simultaneously enhance customer utility, banks are increasingly focusing on their individual core capabilities while exploring different sourcing options for non-core capabilities. Consequently, they are disaggregating their value chain into independently operable functional units. As communication capabilities reach higher levels of performance and reliability, these functional units are combined across corporate borders, thereby increasing sourcing options and flexibility."

Independent functional units can therefore perform their function both for their owner organization *and* outside firms whish require the given functionality (e.g., [15]). Broadly speaking, such units are called services within the services perspective. Herein we call them modules since we have worked on both services and agents as enablers of pluripotent IS elsewhere [2], [3]. While modules can evidently be much less elaborate than entire organizational units (as fully automated web services usually are), the same broad principles apply: both are self-describing and self-contained units designed to execute a well-delimited task, and have standardized interfaces to the outside environment. It is not difficult to picture an efficient approach given increased modularity, openness, distribution, and interoperability, e.g., to loan provision. First, the loan institution would use separate services for each of the various activities involved in loan provision (e.g., information gathering and assembly, credit analysis, application evaluation, risk evaluation, customer service, customer administration, refinancing, etc.). Second, there would be competing modules for each of the tasks, so that the loan institution can choose for each task the module which suits it best. Third, as new modules become available, the loan institution would revise the composition of its value chain, using new modules instead of those previously employed. Ideally, the process of selecting and coordinating the modules would be automated, so that the loan institution continually uses only modules that "best" fit its requirements in terms of, e.g., efficiency, privacy, security, and of the intended offering to customers who apply for loans.

While loans cannot be provided in the described manner at the time of writing, the interest in doing so is apparent: (a) individual modules may achieve economies of scale by working for more than one loan institution; (b) the loan institution may benefit from the focus (i.e., specialization) of individual modules (that is, their providers) on their respective tasks; (c) the loan institution may economize on switching costs due to interoperability. Furthermore, if the module selection process is automated, the loan institution may further economize on (d) human costs, for it will not invest resources in manually selecting appropriate modules, and (e) time needed to identify and switch to better modules.

If we abstract from the hypothesized loan provision example, realizing the efficiencies (a)–(e) requires solving the following problem:

> Given a large number of competing, interoperable modules which describe their offerings using standardized and machine-understandable notations:
> 1) How do we express the requirements that need to be satisfied through the use of some of these modules?
> 2) How do we use the given requirements to automatically select at all times the modules that can best satisfy the said expectations?

The first question above is directly related to the earlier discussion of requirements variability (§II-B), whereas the second issue relates to the problem of selecting appropriate modules given variable module availability and the possibility for new modules to become available (§II-A).

In summary, pluripotent IS arise from the new possibilities offered by increasing modularity, openness, distribution, and interoperability, either through service- or agent-orientation. We argue below that pluripotent IS are a solution to a particular subclass of the service composition problem in service-oriented computing, or the task allocation problem in agent-oriented computing.

III. Composition Problem in Pluripotent IS

One way in which the satisfaction of a request submitted to a pluripotent IS proceeds is shown in Figure 1. An arbitrary[3] idle composer in the pluripotent IS receives a request (see, plate a in Figure 1), described in terms of functional (drawn as a process in this example) and quality requirements (b). The composer interprets the request, identifies modules appropriate w.r.t. functional and quality requirements (c), allocates modules to tasks in the process, and coordinates the execution of these modules (d). As new modules appear (e), the composer continues to execute the same composition (f) which gives a certain level of success (g), whereby the success rate designates the proportion of request executions which satisfy all requirements laid out in the request. In parallel, a duplicate composer explores new compositions by randomly choosing new modules among those that appeared (h), and allocates them to appropriate tasks in the process (i). New composition that gives a higher success rate is then used instead of the old composition (j). Illustrative success rate data in Figure 1 comes from experiments we presented elsewhere [2], [3].

The described procedure is not unlike that of service composition in service-oriented computing or task allocation in agent-oriented computing. Above, we assume that the process to execute and the quality considerations are given in the request. We used these assumptions elsewhere [2], [3] to focus on the problem of how the composer ought to revise prior compositions to account for new modules appearing and

---

[3]The architecture we used more elaborate than described herein. Namely, we do not allocate requests always to arbitrary composers, but have specialized composers for requests that arrive above some threshold frequency. We do not discuss this further here; the interested reader is referred to [2].

other modules becoming unavailable. Given that the problem is similar to that of service composition, we now consider whether alternative approaches may be relevant for enabling pluripotent IS.

Looking at alternative approaches to composition, we observe that the choice of assumptions about the characteristics of the IS and of the requests influence the solution to the composition problem as follows:

- *Are only functional (A) or both functional and non-functional (B) criteria used in selecting modules for the composition?* When selecting relevant modules from the pool of available modules, the composer first needs to identify those modules that can perform the needed functionality. If only functional criteria are used (A), there is no apparent way of comparing competing modules which can all perform the same functionality (e.g., [16]). To add nonfunctional criteria (B), it is necessary to assume a QoS ontology for modules (e.g., [17]), which the providers use to advertise modules' nonfunctional characteristics, and over which the composer can compare alternative modules.
- *Is the process to execute known (C) or not (D) in the request?* If unknown, a description of the goal state is usually given. From there on, planning can be applied to identify, given a set of modules whose functionality is known, the sequence of modules whose execution leads to the goal state. If the process is known, the compositon problem amounts to allocating the tasks to modules that can execute the given tasks.
- *Does the composer rely on advertised (E) or observed (F) values of nonfunctional characteristics of individual modules?* When several modules provide the same functionality, the composer compares the modules based on their nonfunctional characteristics. Service providers advertise the values for their modules' nonfunctional characteristics. If the composer rates the modules based on advertised values (E), it is assumed that the modules attain advertised performance levels in all executions. Otherwise, the composer compares modules based on observed performance in prior compositions (F), consequently accounting for discrepancies between advertised and actual behavior.
- *Does the composer revise compositions as new modules become available?* No revision (G) implies that each composition is optimal, and thus that there can not be new modules that perform better than those already available. If compositions are revised (H), the composer may proceed to replan the composition to explore new compositions based on newly available modules.
- *Is the set of all functional and nonfunctional characteristics for modules known regardless of what modules are available?* If yes (I), any new module appearing in the system necessarily performs some functionality that is known, and over which the users can express requirements. Same applies for nonfunctional characteristics— all are known in advance, so that any new module

advertises its nonfunctional characteristics over those already known. On the Semantic Web, this is equivalent to say that the ontologies of functional and nonfunctional characteristics are known in advance. This is somewhat unrealistic, for it would imply that no module provider can offer new functionality through its modules. If unknown (J), users need to be informed of new functionality that becomes available as new modules appear: ontologies are not entirely predefined, but need to be updated at runtime.

Prominent results in module composition rely on the Golog [18] logic programming language. Starting from generic process descriptions and user preferences, compositions are planned to satisfy preferences [19], [20], [21]. In this respect, they rely on functional criteria in selecting modules (A), plan compositions from goals and preferences (D)[4], rely on advetised functional characteristics (E), and either do not consider the revision problem, or when they do (H), the composition is replanned [22], [23]. The functional ontology is assumed predefined for the problem of new functionalities is not addressed (I). Another noted approach [24], [25], [26] combines the features (A, D, E, G, I); assuming that individual modules are described as stateful processes with BPEL4WS [27], a goal state is partially specified in temporal logic, a plan that satisfies the goal is synthesized through planning via symbolic model checking, and the plan is translated back into BPEL4WS so that it can be submitted for execution. For other related efforts, the reader is referred to discussions in the relevant literature (in particular, see, [24], [21], [3]). Within this literature, we studied automated composition under the feature set (B, C, F, H, I) and proposed a novel reinforcement learning algorithm which allows us to account for both functional and nonfunctional considerations when allocating modules to tasks given in a process model. The algorithm takes nonfunctional criteria to optimize, a process model, a set of hard constraints on nonfunctional characteristics of modules, then learns the optimal allocation of modules to process steps (i.e., creates a composition), and continually explores allocations of newly available modules to process steps (thus revising compositions as new modules appear). The approach that we advocated suffers in that it requires a defined process model. In contrast then to the mentioned notable composition approaches, we achieved advanced adaptability and quality-orientation (by allowing nonfunctional criteria to be taken into account during composition), but lacked where these approaches are strongest: in plan synthesis based on functional requirements.

To enable pluripotent IS, our long term aim is to enable composition that bears the feature set (B, D, F, H, J). First, we can take both functional and nonfunctional requirements as module selection criteria (B), so that we can account for quality of service considerations in the system. Second, very few assumptions need be made as to the expertise of the users specifying the request, so that no predefined process

---

[4]Though some generic procedure for how to achieve the goal is usually given—see, e.g., [21].
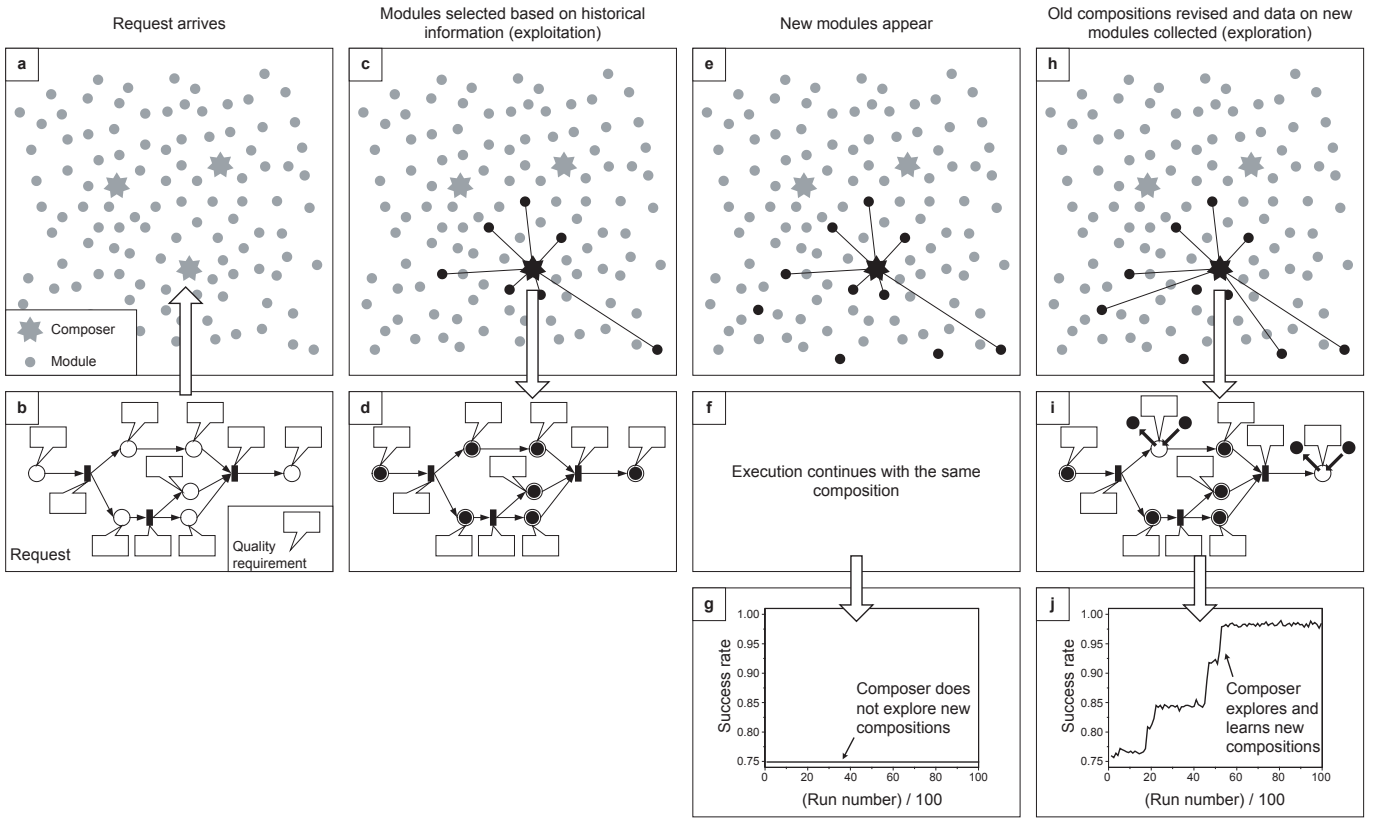
Fig. 1. An illustration of composition in pluripotent IS.

need be assumed (D). Third, uncertainty makes it difficult to assume that modules perform as advertised—it is better to base subsequent compositions on the observed prior performance of modules rather than their advertised performances (F). Fourth, compositions are continually revised so that users always obtain optimal solutions to their requests (H). Finally, we assume that ontologies of functional and nonfunctional characteristics are only partly known, and need to be revised as modules with new characteristics appear (J). Intuitively, this is a particularly attractive aim, which fits well the loan problem we outlined earlier. The loan institution would be able to specify requests rich in functional and nonfunctional criteria for composition; it would not need to specify the detailed process to execute, leaving it instead to its internal or external modules and the composer to plan; inside or outside partners would be selected (and compensated) based on observed and not advertised behavior; the set of involved parties would be constantly revised for optimal results; finally, the loan institution remains open to novel approaches to providing and managing loans.

The (B, D, F, H, J) combination of features leads to the following formulation of the research problem that is to be resolved if pluripotent IS are to be of interest in industry:

The pluripotent IS problem.

Given:

- a set $\mathbf{W}(t)$ of *modules* available at time t.
- a set $\mathbf{W}_\infty$ of all possible modules. At time t, we know

only the distinct modules from $\mathbf{W}_\infty$ that have been available up to t.
- a set $\mathbf{C}(t)$ of *composers* available at time t.
- a set $\mathbf{R}(t)$ of *requests* submitted to composers at time t.
- a *cumulative task domain ontology* $\mathcal{O}^{\mathbf{T}}(\bar{t})$ that defines functionalities offered by all modules that have been available up to and including time t (hence, 'cumulative'). We call individual functionalities '*tasks*'. $\bar{t}$ is a sequence ending with, and including t.
- a *full task ontology* $\mathcal{O}^{\mathbf{T}}_\infty$ of all possible tasks that modules can offer. At any time t, we only know the part $\mathcal{O}^{\mathbf{T}}(\bar{t})$ of $\mathcal{O}^{\mathbf{T}}_\infty$.
- a *task advertising function* $\mathcal{M}^{\mathbf{T}} : \mathbf{W}_\infty \longrightarrow \wp\ddot{\mathcal{O}}^{\mathbf{T}}_\infty$ which is a total function mapping each individual module to one or more tasks that the given module can execute. We use $\ddot{\mathcal{O}}^{\mathbf{T}}_\infty$ to denote the set of all distinct entities extracted from the ontology $\mathcal{O}^{\mathbf{T}}_\infty$.
- a *cumulative quality domain ontology* $\mathcal{O}^{\mathbf{Q}}(\bar{t})$ that defines nonfunctional characteristics of all modules that have been available up to and including time t. We call individual nonfunctional characteristics *qualities*.
- a *full quality ontology* $\mathcal{O}^{\mathbf{Q}}_\infty$ of all possible qualities that can characterize all possible modules. At any time t, we only know the part $\mathcal{O}^{\mathbf{Q}}(\bar{t})$ of $\mathcal{O}^{\mathbf{Q}}_\infty$.
- a *quality advertising function* $\mathcal{M}^{\mathbf{Q}} : \mathbf{W}_\infty \longrightarrow \wp\ddot{\mathcal{O}}^{\mathbf{Q}}_\infty$ which is a total function mapping individual modules

to sets of qualities from the cumulative nonfunctional domain ontology. We use $\ddot{\mathcal{O}}^{\mathbf{Q}}_{\infty}$ to denote the set of all distinct entities extracted from the ontology $\mathcal{O}^{\mathbf{Q}}(\bar{t})$.

such that:

- for each module $\mathbf{w} \in \mathbf{W}(t)$, functional $\mathcal{M}^{\mathbf{T}}(\mathbf{w})$ (i.e., tasks) and quality characteristics $\mathcal{M}^{\mathbf{Q}}(\mathbf{w})$ are advertised.
- each request $\mathbf{r} \in \mathbf{R}(t)$, $\mathbf{r} \equiv \langle \mathbf{r}^{\mathbf{T}}, \mathbf{r}^{\mathbf{Q}}, \mathbf{r}^{\mathbf{O}} \rangle$, defines constraints $\mathbf{r}^{\mathbf{T}}$ on tasks from $\mathcal{O}^{\mathbf{T}}(\bar{t})$, constraints $\mathbf{r}^{\mathbf{Q}}$ on qualities from $\mathcal{O}^{\mathbf{Q}}(\bar{t})$, and identifies the qualities $\mathbf{r}^{\mathbf{O}}$ from $\mathcal{O}^{\mathbf{Q}}(\bar{t})$ to optimize.
- the cumulative task domain ontology is updated at each time period t to include tasks that are advertised for all $\mathbf{W}(t)$.
- the cumulative quality domain ontology is updated at each t to include all qualities that are advertised for all $\mathbf{W}(t)$.

Find:

1) a procedure to elicit and specify functional and nonfunctional requirements.
2) a procedure to transform requirements into requests, i.e., to obtain $\mathbf{r}^{\mathbf{T}}$, $\mathbf{r}^{\mathbf{Q}}$, and $\mathbf{r}^{\mathbf{O}}$ for each request $\mathbf{r} \in \mathbf{R}(t)$.
3) a procedure to update the specification of requirements to inform the users of the extent to which their prior requests have been satisfied and of the new requirements that the pluripotent IS can satisfy.
4) a procedure $\mathcal{P}$, which given any particular request $\mathbf{r}$, returns a sequence of tasks $\mathcal{P}(\mathbf{r})$ to execute in order to satisfy $\mathbf{r}^{\mathbf{T}}$.
5) a procedure $\mathcal{A}$, which given $\mathcal{P}(\mathbf{r})$ and the available modules $\mathbf{W}(t)$, returns an *allocation* $\mathcal{A}(\mathcal{P}(\mathbf{r}), \mathbf{W}(t))$. The allocation indicates which module in $\mathbf{W}(t)$ is to execute what task in $\mathcal{P}(\mathbf{r})$ in order to satisfy $\mathbf{r}^{\mathbf{Q}}$ and optimize $\mathbf{r}^{\mathbf{O}}$.
6) a procedure $\mathcal{R}$ which, at some subsequent time t', $\mathbf{W}(t') \neq \mathbf{W}(t)$, explores alternative allocations and returns $\mathcal{A}(\mathcal{P}(\mathbf{r}), \mathbf{W}(t'))$ which optimizes $\mathbf{r}^{\mathbf{O}}$.

Issues 1–3 are problems of requirements engineering, that is, concern the elicitation, specification, and analysis of requirements. Issue 2 concerns mainly the relationship between requirements and capabilities that realize them—one way to look at this is through traces that can be established between requirements and system behaviors. Concerns 4–6 are problems of planning. Allowed expressivity of requirements affects the choice of the planning approach relevant for issues 4–6; that is, solutions to the various issues are intertwined for the choices in one affect the choices that can be made in resolving others. We discuss in the remainder our contributions to the resolution of the above problem.

IV. Engineering Requirements for Pluripotent IS

There are no requirements engineering (RE) methodologies specific to pluripotent IS. In such a setting, we have studied whether pluripotency places sufficient difficulties on established RE methodologies to warrant a departure from accepted approaches. Any established RE methodology, such as, e.g.,

KAOS [28] and Tropos [29] would start with early and late requirements analyses to better understand the organizational setting, where dependencies between the module providers and end users would be identified, along with the goals, resources, and tasks of these various parties. Architectural design would ensue to define the sub-systems and their interconnections in terms of data, control, and other dependencies. Finally, detailed design would result in an extensive behavioral specification of all system components. While other methodologies, such as KAOS [28] involve a somewhat different approach, all move from high-level requirements into detailed behavioral specifications. This established approach is limited for pluripotent IS for several reasons. First, openness to new modules guarantees that the requirements engineer does not know all potential behaviors during the RE phase of the system development process. The problem defined earlier states that only the cumulative task ontology $\mathcal{O}^{\mathbf{T}}(\bar{t})$ (instead of the full) is known, so that the capabilities of the system cannot be fully understood nor documented before deployment. Detailed design is thus limited to the design of composers, not individual modules. Second, it cannot be reasonably expected for all modules to be always available nor their quality of service to be perfectly stable. Setting precise levels at which quality metrics need to be satisfied and doing this before deployment is therefore unrealistic. Moreover, new modules may carry quality metrics that have not been encountered with modules previously used at runtime. Indeed, only the cumulative ontology $\mathcal{O}^{\mathbf{Q}}(\bar{t})$ of nonfunctional characteristics is known at any given time. Overall, applying the established RE process to pluripotent IS can be problematic if it fixes too strongly the requirements at the outset, thus making it impossible for composers to consider new modules at runtime.

One response which benefits from the established advances in RE, yet accommodates the variability in how (in terms of tasks) and how well (in terms of quality) requirements are satisfied, lies in knowing to which extent (i.e., to what level of detail) to specify requirments for a pluripotent IS during development, and then to update the requirements specification at runtime to reflect the actual behavior of the system. The requirements specification thus becomes 'dynamic' in that it is continually updated at runtime. Another approach is to explore entirely new processes for the RE of pluripotent IS. We have studied the first approach and defined a semi-automatic algorithm used to update a requirements specification produced using an established RE methodology, such as Tropos or KAOS. The algorithm builds defeasible traces between fragments of a requirements specification and fragments of requests. The traces can be seen as mappings between the statement of stakeholders' requirements and the machine-understandable format of these same requirements (i.e., their corresponding specification in a request). Given a requirement fragment, the algorithm provides guidelines on manually converting the said requirement into a fragment of the request, depending on the content of the requirements fragment (i.e., whether it is a functional or nonfunctional

requirement, a preference, or a priority) and of the available cumulative task and quality ontologies. The conversion gives rise to a defeasible trace between the requirement fragment and the request fragment. The trace is recorded in a repository and its consistency is checked against already available traces. Once recorded, each particular trace performs two functions: (i) given a requirement fragment that corresponds to the trace at any point of runtime, the trace is used to automatically generate the corresponding request fragment, thus helping to write requests from requirements; (ii) when a new module becomes available, described in the same language in which the requests are written, the trace is used to convert the module description into requirements, and therefore advertise new capabilities in a more convenient format to the users. The algorithm thus allows users to learn about newly available functionality as the new capabilities are described in terms of potential requirements. One salient feature of the traces is that they are defeasible—that is, can be revised at runtime; this was needed as the task and quality ontologies are cumulative, so that some traces may become inappropriate as the underlying ontologies evolve. Technical details of the algorithm are presented elsewhere [4], [5].

In its current form, the algorithm and the overall approach are not ready for industrial use: considerable human intervention is required when defining traces. The approach did, however, point to one relevant direction for research in RE, namely, the representation of and reasoning about preferences over requirements, and priorities between preferences. Preferences are of clear relevance in pluripotent IS—the user states what is ideally to be satisfied by the composer, but also makes explicit alternative and still acceptable degrees of satisfaction. Given that the pool of available modules varies, preferences are necessary to avoid either idealistic requirements (so that the composer cannot obtain a satisfactory composition given a pool of modules) and/or much too restrictive requirements. Restrictive requirements arise from stakeholders' pessimistic expectations, whereby it happens that the composer actually can satisfy requirements to a more desirable extent, but is limited by the fixed restrictive requirements. Preferences that cannot be simultaneously satisfied are conflicting. Priorities are defined over conflicting preferences; a priority order gives an order of importance over preferences. Once the composer encounters a conflict of preferences, it seeks an optimal composition which optimizes the most important preferences first. Hence, priorities help in resolving trade offs at runtime. We recently finished a first report which explores the relationships between the concepts of preference and priority, and established categories in RE (e.g., goals, softgoals, constraints, etc.) [1], [6], [7]. The resulting conceptual framework will hopefully act as a starting point for developing requirements elicitation and specification methods for pluripotent IS.

In summary, our efforts on the RE of pluripotent IS involved mainly the definition and testing of concepts and techniques which are grounded and reuse established results in RE. In light of the problem definition given earlier, we have worked mainly on issues 2 and 3. Work on issue 1 will benefit from results in the modeling and analysis of preferences and priorities in artificial intelligence (e.g., [30], [31]).

V. CONTINUALLY LEARNING OPTIMAL COMPOSITIONS

In the context of the problem defined earlier, our efforts focused on issues 5 and 6, that is, the identification of the appropriate composition given a request and a pool of available modules, and the revision of the composition to account for new modules or the unavailability of previously used modules. We have therefore assumed that the sequence of tasks to execute in order to fulfil the request is known. The problem we have tackled is one of finding out how to allocate tasks to modules (equivalently: how to assign modules to tasks) and continually revise the allocation.

Difficulty in defining a task allocation procedure for composers in pluripotent IS arises from the necessity to account for the variable availability of modules, the availability of many competing modules, and the non-determinism of module executions (i.e., we cannot be sure whether the execution of a module will go on as expected). Within the present discussion, a task allocation procedure which acknowledges and is robust with regards to these considerations ought to correspond to the feature set (B, C, F, H, J).

To enable this feature set, we advocate that module compositions optimal with regards to a set of criteria need to be learned at runtime and revised as new modules appear and availability of old modules changes, whereby the learning should be based on observed module performance, and not the performance advertised by the module providers. To enable such learning, an allocation procedure is needed which both *exploits* the observed past performance of modules, and *explores* new composition options to avoid excessive reliance on past data. To this aim, we suggested elsewhere the *Multi-Criteria Randomized Reinforcement Learning* (MCRRL) approach to module composition [2], [3]. MCRRL integrates two components:

- A generic request model to describe the process to execute by the module composition and the criteria and constraints to meet when executing it. The request model indicates the kind of information that the algorithm expects from the user when allocating process tasks to modules. In MCRRL, the process model is specified in the form of a statechart. Statecharts provide an expressive formalism for describing processes, have well defined syntax and semantics so tools can be applied for verification and silmulation purposes, and incorporate flow constructs used in available process modeling languages. These languages (e.g., BPMN [32]) can thus be used to specify processes in MCRRL.

- A reinforcement learning algorithm, called Randomized Reinforcement Algorithm (RRL), to select the modules that are to perform tasks specified in the request. The algorithm decides on the modules to select among competing modules, and this based on multiple criteria (including various quality of service parameters, deadline,

reputation, cost, and user preferences), while both exploiting available module performance data and exploring new composition options.

In contrast to comparable related work (mentioned earlier, §III), MCRRL uses reinforcement learning (RL) to allocate tasks to modules. RL (see, e.g., [33] for an introduction) is a particularly attractive approach with regards to the above problem. RL is a collection of methods for approximating optimal solutions to stochastic sequential decision problems. An RL system does not require a teacher to specify correct actions. Instead, the learning agent (here, the composer) tries different actions and observes the consequences to determine which are the best, given a set of criteria to obey and parameters to optimize. Each action is an allocation of a task to a module, whereby the module executes the allocated task whereby the transition occurs, and the next task then needs to be allocated. After the transition, the composer receives a positive or negative 'reward', thus reinforcing or weakening the tendency to allocate the previously allocated task to the module which just executed the task. The composer then proceeds to allocate the following task. One advantage of RL over, e.g., queuing-theoretic algorithms (e.g., [34]), is that the procedure for allocating modules to tasks is continually rebuilt at runtime: i.e., the composition procedure changes as the observed outcomes of prior composition choices become available. The composer tries various allocations of tasks to modules, and learns from the consequences of each allocation. Another advantage is that RL does not require an explicit and detailed model of either the computing system whose operation it manages, nor of the external process that generates process model. Finally, being grounded in Markov Decision Processes, the RL is a sequential decision theory that properly treats the possibility that a decision may have delayed consequences, so that the RL can outperform alternative approaches that treat such cases only approximately, ignore them entirely, or cast decisions as a series of unrelated optimizations.

One challenge in RL is the tradeoff between *exploration* and *exploitation*. Exploration aims to try new ways of solving the problem, while exploitation aims to capitalize on already well-established solutions. Exploration is especially relevant when the environment is changing: good solutions can deteriorate and better solutions can appear over time. In module composition, exploitation consists of learning optimal allocations of tasks to modules, and systematically reusing learned allocations. Without exploration, the composer will not consider allocations different than those which proved optimal in the past. This is not desirable, since in absence of exploration, the composer is unaware of changes in the availability of WS and appearance of new modules, so that the performance at which requests are fulfilled inevitably deteriorates over time in an open and distributed system.

If RL is applied to task allocation, the exploration/exploitation issue can be addressed by periodically readjusting the policy for choosing task allocations and re-exploring up-to-now suboptimal execution paths [35], [33]. Such a strategy is, however, suboptimal because it does not account for exploration. Our algorithm, inituially introduced in [36] has been subsequently [2], [3] adapted to task allocation to allow the assignment of tasks to modules while: (i) optimizing criteria, (ii) satisfying hard constraints, (iii) learning about the performance of new modules so as to continually adjust task allocation, (iv) exploring new options in task allocation, and (v) accomodating concurrent allocations. The exploration rate is quantified with the Shannon entropy associated to the probability distribution of allocating a task to a module. This permits the continual measurement and control of exploration. Further technical details of the algorithm have been presented elsewhere [2], [3].

Apart from not accommodating situations which the precise sequence of tasks is unknown, the approach briefly introduced above requires significant computing resources for any realistic use. Numerous iterations are needed to learn the most appropriate composition, so that real time performance is still not satisfactory. Optimization to specific domains through guided exploration is one way to improve performance in industrial settings.

VI. DISCUSSION AND CONCLUSION

We have argued above (§III) that the appropriate set of features is (B, D, F, H, J) for composers in a pluripotent IS. We have presented elsewhere [1], [2], [3], [4], [5], [6], [7] fragments of the solutions which are oriented towards enabling the said feature set. We have shown in this paper how these various techniques and concepts combine to enable the engineering of pluripotent IS. More precisely, our efforts resulted in a framework, outlined above, which enables the building of systems with the feature set (B, C, F, H, J), whereby we have suggested a rich ontology for the functional and nonfunctional requirements (which concerns the feature B). Returning to the loan provision example (§II-C), the approach outlined above allows the engineering of the IS that can support the flexible loan provision we have described as a desirable scenario.

Pluripotent IS are not intended to replace or compete with available paradigms in IS engineering. As we have argued throughout the paper, pluripotent IS rely on available paradigms such as service- and agent-oriented computing, and are relevant when particular conditions hold. Namely, developers of the system cannot anticipate precisely all requirements that the stakeholders may have on the system at runtime, the system relies on modules developed by providers outside the system's development team, and when the resulting system is intended to cater to wide ranging requirements (i.e., it can be put to serve various purposes).

The research reported here is still in progress. We have highlighted above the main limitations of the overall approach. Two limitations stand out. First, the technique for the continual update of rich requirements specifications still involves considerable manual effort. Second, the composition algorithm (MCRRL) is costly in computational resources. We are working to improve our proposal in both areas. Namely,

automation of the continual requirements update process is being explored through the reuse of results in defeasible logic programming. Automation of preference and priority elicitation and consistency checking is also a concern, and will benefit (as mentioned earlier) from available results in artificial intelligence. As our evaluations were mostly grounded in controlled simulation settings, we are currently working on an open software platform which will enable the simulation and testing of a pluripotent IS in more realistic settings. Work is currently being performed to extend the approach outlined herein to allow the processes not to be known—that is, to enable process discovery given a goal state only. After additional experience is gained on pluripotent IS, we intend to work on tools to assist the engineering of pluripotency into IS and the evolution of available IS towards a pluripotent architecture.

References

[1] Jureta, I.J., Faulkner, S., Schobbens, P.Y.: A more expressive softgoal conceptualization for quality requirements analysis. In: Proceedings of the 25th International Conference on Conceptual Modelling (ER'06). (2006)

[2] Jureta, I.J., Faulkner, S., Achbany, Y., Saerens, M.: Dynamic task allocation wihin an open service-oriented mas architecture. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agents Systems (AAMAS'07). (2007)

[3] Jureta, I.J., Faulkner, S., Achbany, Y., Saerens, M.: Dynamic web service composition within a service-oriented architecture. In: Proceedings of the International Conference on Web Services (ICWS'07). (2007)

[4] Jureta, I.J., Faulkner, S., Thiran, P.: Dynamic requirements specification for adaptable and open service systems. In: Proceedings of the 2007 International Requirements Engineering Conference (RE'07). (2007)

[5] Jureta, I.J., Faulkner, S., Thiran, P.: Dynamic requirements specification for adaptable and open service-oriented systems. In: Proceedings of the 2007 International Conference on Service-Oriented Computing (ICSOC'07). (2007)

[6] Jureta, I., Faulkner, S., Schobbens, P.Y.: Achieving, satisficing, excelling. In: Proceedings of the 1st International Workshop on Requirements, Intentions and Goals in Conceptual Modeling (RIGiM'07). (2007)

[7] Jureta, I.J., Mylopoulos, J., Faulkner, S., Schobbens, P.Y.: Core ontology for requirements engineering. Technical report, Information Management Research Unit, University of Namur (2007)

[8] Tennenhouse, D.: Proactive computing. Commun. ACM **43**(5) (2000) 43–50

[9] Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer **36**(1) (2003) 41–50

[10] Yue, K.: What does it mean to say that a specification is complete? In: Proceedings of the International Workshop on Software Specification and Design (IWSSD'87). (1987)

[11] van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings of the International Symposium on Requirements Engineering (RE'01). (2001)

[12] MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R.: Reference model for service oriented architecture 1.0 (committee specification 1). Technical report, Organization for the Advancement of Structured Information Standards (OASIS) (2006)

[13] Homann, U., Rill, M., Wimmer, A.: Flexible value structures in banking. Commun. ACM **47**(5) (2004) 34–36

[14] Baskerville, R., Cavallari, M., Hjort-Madsen, K., Pries-Heje, J., Sorrentino, M., Virili, F.: Extensible architectures: The strategic value of service-oriented architecture in banking. In: ECIS. (2005)

[15] III, J.H., Singer, M.: Unbundling the corporation. Harvard Business Review (1999)

[16] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., Sheng, Q.Z.: Quality driven web services composition. In: Proceedings of the International World Wide Web Conference (WWW'03). (2003)

[17] D'Ambrogio, A.: A model-driven wsdl extension for describing the qos of web services. In: Proceedings of the International Conference on Web Services (ICWS'06). (2006)

[18] Levesque, H.J., Reiter, R., Lesperance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. Journal of Logic Programming **31**(1-3) (1997) 59–83

[19] McIlraith, S., Son, T.C.: Adapting golog for composition of semantic web services. In: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR'02). (2002)

[20] Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: Proceedings of the International Conference on the World Wide Web (WWW 2002). (2002)

[21] Sohrabi, S., Prokoshyna, N., McIlraith, S.: Web service composition via generic procedures and customizing user preferences. In: Fifth International Semantic Web Conference (ISWC2006). (2006) 597–611

[22] Lespérance, Y., Ng, H.K.: Integrating planning into reactive high-level robot programs. In: Proceedings of the Second International Cognitive Robotics Workshop. (2000)

[23] Martínez, E., Lespérance, Y.: Web service composition as a planning task: Experiments using knowledge-based planning. In: Proceedinds of the ICAPS-2004 Workshop on Planning and Scheduling for Web and Grid Services. (2004)

[24] Pistore, M., Traverso, P., Bertoli, P., Marconi, A.: Automated synthesis of composite bpel4ws web services. In: ICWS, IEEE Computer Society (2005) 293–301

[25] Pistore, M., Traverso, P., Bertoli, P.: Automated composition of web services by planning in asynchronous domains. In Biundo, S., Myers, K.L., Rajan, K., eds.: ICAPS, AAAI (2005) 2–11

[26] Kazhamiakin, R., Pandya, P.K., Pistore, M.: Representation, verification, and computation of timed properties in web. In: ICWS, IEEE Computer Society (2006) 497–504

[27] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services version 1.1. Technical report, BEA, IBM, Microsoft, SAP AG and Siebel Systems (2003)

[28] Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming **20** (1993)

[29] Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the tropos project. Information Systems **27**(6) (2002) 365–389

[30] Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. J. Artif. Intell. Res. (JAIR) **21** (2004) 135–191

[31] Brafman, R.I., Domshlak, C., Shimony, S.E.: On graphical modeling of preference and importance. J. Artif. Intell. Res. (JAIR) **25** (2006) 389–424

[32] Initiative, O.M.G..B.P.M.: Business process modeling notation specification (final adopted specification dtc/06-02-01). Technical report, Object Management Group (2006)

[33] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)

[34] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., Tantawi, A.: An analytical model for multi-tier internet services and its applications. In: Proceedings of SIGMETRICS-05. (2005)

[35] Mitchell, T.M.: Machine learning. McGraw-Hill (1997)

[36] Achbany, Y., Fouss, F., Yen, L., Pirotte, A., Saerens, M.: Optimal tuning of continual online exploration in reinforcement learning. In: Proceedings of the International Conference on Artificial Neural Networks (ICANN'06). (2006)