A new deliberation mechanism for Service-Oriented Operating Systems

Javier Palanca, Martí Navarro, Ana García-Fornés, Vicente J. Julián
Universitat Politècnica de València
Camino de Vera s/n, 46022, Valencia (Spain)
{jpalanca, mnavarro, agarcia, vinglada}@dsic.upv.es

Abstract—Cloud and Service-Oriented Computing have become successful paradigms to use all the power that the network has brought to Operating Systems. In particular, the multi-agent systems paradigm and service-oriented computing offers new ideas for the development of more intelligent and effective OS’s, which would benefit the end-user due to the advantages of both technologies. The benefits that we present in this work are how to improve the security in transactions between clients and providers over the cloud and how these transactions can be done taking into account temporal constraints. This is done either to obtain a result before a deadline, or simply to improve the quality of the result. In this work we present a deliberation engine for an OS execution model based on goals with temporal and security considerations.

Keywords—services; multi-agent systems; operating systems;

I. INTRODUCTION

In recent years, the complexity of the developed software has grown so huge. Also, the need to develop large amounts of software as quickly as possible due to industry requirements has lead to discover that traditional paradigms of software development are not enough to create complex software. That is why there is a constant work on new paradigms, to improve the level of abstraction needed to develop increasingly complex applications with higher reliability, security and performance. Among these paradigms, we can highlight the Service-Oriented Computing paradigm and Multi-Agent Systems.

Service-Oriented Computing (SOC)[1] is a paradigm where the fundamental component for developing applications is the service. Using single services or service compositions it is possible to achieve solutions to problems in a decentralized manner with a high degree of adaptability. This paradigm, coupled with the Cloud Computing one, is becoming very important today because both paradigms allow us to develop applications based on platform-agnostic, distributed and low-cost computational elements. The use of SOC in multi-agent systems is endorsed by the proposal of achieving the agent goals by means of the invocation and composition of a set of services that are available within the multi-agent system.

Dickinson and Wooldridge discuss at [2] different ways in which to consider the relationship between multi-agent systems and service architectures. As is summarized in that work, some authors propose that there is no conceptual distinction between agents and services: both are active building blocks in a loosely coupled architecture [3]. Another approach considers a bi-directional integration where agents and services interoperate by communicating one to each other [4]. Finally, a third approach considers that agents are the software components that invoke services [5], mediating between services and users.

Since agents are intelligent entities and have social capabilities, they fit properly in our service-oriented framework. This framework uses goals as the main element to direct the execution of system entities. Goals activate the search for a composition of actions that perform the needs expressed by the goal using a deliberative process. This paradigm is known as Goal-Oriented Computing (GOC). This framework, presented in [6], is the base of the present work. This approach is based on finding solutions to problems through composition and execution of various services offered by different agents. This functionality should be provided to agents through a specific framework that supports service composition and their subsequent execution. Agents are providers and consumers of services in this framework, where agents use their social capabilities to find a way to fulfill their own goals. This functionality should be provided to agents through a specific architecture that supports service composition and their subsequent execution.

The multi-agent system paradigm consists of very high level abstractions (agent, message, ontology ....). These abstractions have little correlation with the execution support provided by classical operating systems. This leads us to have a very high stratification in the system, where previously it was necessary to introduce layers that support these higher level abstractions. These abstractions are finally converted, by means of more or less artificial changes, in the low-level abstractions offered by the operating system [7].

The Goal-Oriented Computing paradigm is designed to help the creation of applications by reducing the implementation of functionality to services. These self-contained components perform atomic actions that can be useful to achieve a more complex application development. The Goal-Oriented Computing paradigm simplifies the development of applications. In this paradigm the user expresses his goals and the system helps him to achieve them by means of service composition. But this paradigm can also be useful for the operating system design itself. In this way, the operating system can express its own goals and, even more important, can provide its main functionality as services that can be included in the solutions offered by the services compositions generated to achieve goals.
An important point of a new operating system execution model is taking into account three fundamental design factors in every current operating system: performance, reliability and security. In this OS, we focus on the security factor, inasmuch as it is very important to have a strong security support to provide services between agents, including authentication, intrusion detection and encryption mechanisms. In this work the deliberation engine of the referenced framework is presented. It will provide the necessary support to help to find the set of services which fulfill the agent’s goals, trying to maximize the execution quality and improving the security of these transactions. In this paper quality is defined as a set of values that parametrize a composition of services (reliability, time, trust ....), focusing on time, where the goal is satisfied within some temporal parameters. These parameters comprise on one hand the time needed to generate the service composition and on the other hand, the time required for the execution of the composition. This time frame could be defined by the user or may be calculated by the deliberation engine, taking into account the additional quality parameters like the service reliability. As long as this deliberation engine is part of a full operating system it will be able to take into account more parameters to predict the time frame. These parameters are only accessible from the OS layer, like the system workload.

Another reason to incorporate the paradigm within the operating system design is that, given the nature of the framework, it is very difficult to know the temporal parameters of the provided services. This is why it is necessary to include a component into the deliberation engine which helps predicting these temporal parameters. This component will take into account previous executions of the services and environmental conditions at the instant they were executed. Using this information the deliberation engine can determine which will be the parameters that define its temporal behavior in the execution of the composition. For this reason it has been included within the deliberation engine a planner component based on the CBR methodology (Case-Based Reasoning). The execution cycle of the planner is temporally bounded [8] in order to guarantee a correct service composition within some temporal constraints directed by the operating system. Other contribution of the deliberation engine presented in this work is the commitment manager component. The goal of the commitment manager component is to establish agreements between the service providers and the service applicants. By means of these agreements, the deliberation engine can choose a service composition, taking into account the availability of the service providers and establishing a commitment with the providers to execute the service within a deadline. These deadlines are calculated by the commitment manager based on the current workload and the availability of the resources used by the services. The deliberation engine uses this information to evaluate the quality of the service compositions.

Finally, while some security mechanisms are dependent on the provider agent (like encrypting information), some other are best deployed in the operating system itself (like Host-based and Network-based Intrusion Detectors). The framework presented in this work places special emphasis on security, introducing the negotiation and selection of the providers into the deliberation process of the operating system with a highest security degree.

II. RELATED WORK

There are several focuses on the evolution of operating systems design. They are mainly efficiency, security and reliability. These challenges are always present in the design of operating systems, since they are not satisfactorily solved. Although the hardware keeps getting faster, the operating system should always have the least impact on the resources to leave all the processing power to applications. Especially since the popularization of the internet, it is very important to keep user’s information safe and data integrity. The operating system should provide the mechanisms to keep this security. Finally, it is also important to keep reliability. The operating system should provide mechanisms to avoid errors if possible. When it is not possible to avoid errors the operating system should restrict and contain the error to not affect other components of the system. Finally, if possible, the OS should be fault-tolerant to recover from errors with the least impact. Several studies have focused on improving certain aspects of OS as data access or the input/output (I/O) abstractions, leading to propose new abstractions in this field (file, object, socket, ...[9], [10], [11]. However, the paradigms used to design operating systems (like the file-oriented one) propose no significant progress and are far away from current design trends of software engineering.

There are only a few set of operating systems that are proposing different ways of design. The experimental OS Singularity[12] uses code evaluation to determine whether the code executed in the system is valid or not. This code evaluation is done at compile time using a specific language called Sing#. The complication of this proposal is the static component that the evaluation of code introduces, removing flexibility from the system. This proposal is also tied to the process abstraction. Other interesting operating system design is MINIX. MINIX is one of the most popular microkernel OS still in development. Originally designed by Andrew S. Tanenbaum as a study operating system for his students. Its design was a model for the construction of other Operating Systems, while it has continued its own evolution, reaching the third version of the OS in 2006: MINIX 3[14]. The main objective of the third version of MINIX is reliability, devising for it a self-reparable system. They have followed the design philosophy of microkernel, leaving in protected-mode the minimal functionality and placing in user-mode all the remaining functionality. Thus, the user-mode failures are not critical for the system and also, due to a system called Reincarnation Server, failing processes are self-reparable and can be relaunched in the same state they have failed. Although MINIX3 makes a great effort to be a fault-tolerant operating system, its design is still very similar to old-fashioned operating systems, with a big gap between the current applications design and the paradigm used by the OS. XtreamOS[15], [16] is a Grid Operating System. The development of this system is based on the Linux OS and its objectives are transparency and scalability. Transparency is offered to both the user and
the application, since the great advantage of XtreamOS is still offering a Linux interface despite the availability of certain services and resources distributed on the network. Furthermore, this transparency allows heterogeneity among the classic applications of Linux and those found in the Grid. XtreamOS uses virtual organizations (VO) to encapsulate the services and resources in the Grid. A VO administrator is responsible for its creation, management and completion, whether it is static and dynamic. Again, although the grid orientation is very interesting, the design is attached (for backward compatibility reasons) to a monolithic system, Linux.

There is one important reason why no significant progress has been made in the OS execution model abstractions. It is that these abstractions are closely tied to current hardware. Processors are designed to work optimally with processes. Thereby, when adding improvements to the OS execution model, as well as defining new execution abstractions (as it is proposed in this paper) would be interesting to start thinking about adapting hardware to such abstractions.

III. GOAL-ORIENTED COMPUTING

Goal-Oriented Computing (GOC) is defined as a paradigm where the main element is the goal. Goals express what the user wants to perform. In this paradigm goals are achieved by executing atomic services or services compositions. These services are provided by agents, which also manage goals. The agents that use this paradigm are inspired by the BDI agent model [17], and they incorporate the abstractions of knowledge base, services, goals and plans (which are the service compositions) [18]. A GOC agent $A$ is defined through the following tuple:

$$ A = \{KB, SS, CP, GS\} $$

Where:
- $KB$ represents the agent knowledge base.
- $SS$ represents a set of services offered by the agent. This services are used by the agent to perform its goals, but they can also be offered to other agents to help achieving their own goals.
- $CP$ represents a set of pre-compiled plans provided by the agent to meet its goals.
- $GS$ represents the set of goals that are activated in the agent to be achieved.

Moreover, the Services that we can find in this execution model are OWL-S services[19].

An OWL-S service $S_i \in SS$ is defined by the tuple:

$$ S_i = \{SP, GR, PM\} $$

where $SP$ is the Service Profile, $GR$ the Grounding and $PM$ the Process Model of the service. The Service Profile is used to define what the service does. On the other hand, the Grounding allow us to define how to interact with the service and, finally, the Process Model defines how the service is used.

An OWL-S service Process Model can be either a Composite Process or an Atomic Process. A Composite Process is a set of Atomic Processes (which have no internal structure and run in a single step) with an internal structure built up by Composite and Atomic Processes and a few control constructs (sequence, if-then-else, choice, etc).

This kind of OWL-S service is a well-defined standard which provides this model with enough power to construct all the functionality provided by an agent. The services that make up a plan are the real executable part of a plan. A service $S_i$ is also composed of a PreCondition $P$, a PostCondition $Q$ and a set of Inputs and Outputs. The PreCondition is a prerequisite for the execution of a service. The PostCondition $Q$ is the impact that will drive the execution of the service $S$ and the fulfillment of the goal that was activated by the agent. Both $P$ and $Q$ are defined in the functional aspect of the service profile.

By means of a composition of OWL-S services can be obtained a Composite Process. These compositions include both atomic and composite processes and control structures. Moreover, a Plan is a process model that is composed by one or more Composite Process models (again, including composite services, atomic services and control structs). A Plan defines the way to achieve some results or PostConditions by joining different OWL-S services which can be connected. Composite services or even atomic services can be seen as very simple plans, but we also define a plan as the result of joining different composite services in order to achieve a goal.

To give support to the presented model, a Goal-Oriented Execution architecture has been developed. This architecture is presented in this work as the main element for the execution of goals in an operating system. We have called this element the execution module. An execution module that implements the Goal-Oriented Execution architecture is composed by the necessary components that help agents to fulfill their goals with the highest parameters of quality and security. Some of these components are inside the OS kernel (e.g. the Runtime Engine), because they replace a functionality that is provided by the Operating System or they need some information that is only accessible by the OS itself. Other components (like the Deliberation Engine) are sharing their space between user-space and operating system space. This is because some operations are performed directly by agents, but some other operations need to rely some information in the operating system, in order to be performed with highest security or with a comprehensive understanding of the environment. These components (shown in Figure 1) are:

- **Runtime Engine**: The Runtime Engine takes the plans provided by their planners and manages their execution by transferring the service execution to the OS kernel. It uses distributed services provided by agents in other hosts if necessary.
- **Deliberation Engine**: It is responsible for deciding how and in what order plans are executed. This engine negotiates with the agents which provide a service for a current plan. This engine is permanently running in background and evaluating the goals that are activated in the agents to be achieved and selecting them for its completion. This component interacts concurrently with the Runtime Engine, the Commitment Manager and the On-line Planner.
- **Commitment Manager**: Service provider agents negotiate with the commitment manager their availability and, if so, quality and security parameters like their execution within a time window or the required encryption algorithm in transactions. The security parameters can be defined by both the client or the provider. The Commitment Manager will always try to reach these minimum security parameters by negotiating with all the available agents. However, if there are no required security parameters, the CM will always try to get the best deal for a transaction as secure as possible. To calculate the execution within a temporal bound the agent needs to take into account some points like: (i) the current workload, (ii) the availability of the service at the time of the request and (iii) the availability of the needed hardware and software resources to be able to run. For this work the agent needs the help of the OS. The OS can help the agent to predict if it is going to be able to satisfy the request in the defined temporal bounds, and if so, to establish a commitment with the Deliberation Engine. This functionality is offered by the Commitment Manager.

- **On-line Planner**: it is able to compose new plans on-the-fly. It also repairs and refines running plans. This planner is executed concurrently inside the Deliberation Engine. Its task is to help the agents to reach a goal when the agent has no pre-compiled plans to guide it to the goal completion. This is done by composing or repairing plans. The On-line Planner uses a TB-CBP (Temporal Bounded Case Based Planner) to generate the plans at runtime. It uses past cases from the same service or similar services to generate a plan with a time prediction inside the established temporal bounds.

- **OS Goals Set**: The OS has its own goals to perform the corresponding tasks of an operating system. This set of goals includes all the maintenance tasks and non-critical functionality.

- **OS Knowledge Base**: This is the knowledge that the OS has. The operating system uses this knowledge base to perform their goals by means of the services that can invoke.

- **OS Services Set**: The set of basic services provided by the OS. This set of services is used by the OS to provide the basic low-level functionality to the system agents. It includes all the necessary stuff to manage the system and to access to restricted features only available through the OS for security and stability reasons. Some of these features are the communication of system drivers with the hardware, as well as other features that allow the correct interaction among agents, service providers and the operating system.

- **OS Plan Library**: It provides pre-compiled plans for their execution from a set of goals. This component is created in the design phase of the OS and its motivation is to provide pre-compiled plans for critical goals that cannot wait for a different composition or cannot vary their execution flow due to security and efficiency reasons.

Under this computing paradigm the goals of each agent are sent to the execution module for their achievement. Then the module chooses the appropriate plan to meet each goal. Note that the agent model preserves its desirable features like autonomy and pro-activity since the agent is the component that activates its own goals when it is prepared to achieve them. The module provides the needed resources to help to achieve the goals. Plans may be provided by the agent itself or can be compounded on-line. These plans are a sequence of services offered both locally and remotely by the agents. It is also an agent's capability to share its pre-compiled plans with other agents.

The basic running elements are the services that make the plans. Plans are provided to the module in two different ways: the off-line generation of the plan or the on-line generation of the plan by the On-Line Planner module. Once the plan that meets the active goal is selected, the Runtime Engine activates the services that comprise the selected plan. In Figure 2 is shown the deliberation and execution processes. This figure shows (i) how a goal is selected for execution, (ii) a plan is generated to achieve the selected goal, (iii) the temporal commitment for the execution is done and finally (iv) the plan is executed. If the plan fails, a plan reparation can also be performed. In the following section all this steps and their involved components are showed.

Once the Goal-Oriented Execution model and architecture have been presented, in next section we are going to show the deliberation process that is used to fulfill the agent’s goals within a high quality and safety parameters.

**IV. DELIBERATION ENGINE**

The Deliberation Engine is the brain of the execution module. This component is in charge of analyzing the current active goals and helping their achievement. The Deliberation
Engine is the root node which manages all the main flow of the execution process. Its main task is to get a plan that fits properly with the activated goals. If the plan does not exist, the Deliberation Engine will compose a new plan using its component called On-line Planner. The On-line Planner returns a set of plans that guide the agent to the fulfillment of the goal. The Deliberation Engine uses two classifiers that help the agent to select the most proper plan. Both classifications are performed by the two components included in the Deliberation Engine: the On-line Planner and the Commitment Manager. The On-line Planner makes a first classification using the information retrieved from past executions. A second classification is done by the Commitment Manager. It finds the best providers which offer the required services to complete the plan. This classification establishes commitments with the provider agents to complete the service taking into account security and temporal constraints. Temporal commitments have different parameters like when the service must be run, when will the service end and the probability of finishing the service in that deadline. Security commitments present some constraints about three security concerns: authentication, intrusion detection and encryption. This two components are shown in more detail in Sections IV-A and IV-B. Once a plan is selected, the Deliberation Engine sends it to the Runtime Engine to be run.

Since a goal is activated by an agent until it is achieved, the Deliberation Engine goes through different steps which involve the different components of the execution module. These steps are:

1) Checking if it is possible to activate the goal.
2) Checking if the goal is consistent and there are no conflicts.
3) Asking the On-Line Planner for a set of plans to achieve the goal.
4) Querying the Commitment Manager for temporal and security commitments for each service of the plan.
5) If there is no available commitment, asking the On-Line Planner for a new plan or setting the goal as unreachable.
6) Selecting the best plan from the set of plans using the temporal and security commitments and the information retrieved from past executions.
7) Sending the plan to the Runtime Engine to be executed.
8) If the plan fails, asking the On-Line Planner for a new plan or setting the goal as unreachable.
9) When the plan ends, updating the case-base with the results of the commitments, penalizing or rewarding the providers if necessary.
10) Checking entailed goals and postconditions and setting the goal as reached.

Next the components used by the Deliberation Engine to determine the service composition are presented in more detail.

A. On-line Planner

Within the execution module, the responsible entity for providing plans that fulfill the agents’ goals is the On-Line Planner. This component generates plans composed on-the-fly that achieve the goals that are activated by the agents. This generated plans complement the static pre-compiled plans provided by the agent Plan Library. The On-line Planner is based on a CBP (Case-Based Planning) methodology [20] that has been modified for giving a temporal bounded response. This new model (called Temporal Bounded CBP) is composed by the same phases as the classic CBP, but these phases have been treated to bound their execution time. Thus, the execution time of the service composition process is known and is taken into account when the On-line Planner is building a plan within a maximum time. This paper has not as purpose to introduce in detail the characteristics of a TB-CBP. A comprehensive description of this approach can be found in [8]. Anyway, a general description of the functioning of the TB-CBP on-line planner is shown below.

The case structure used in the TB-CBP is defined as: \(<\text{Postcondition}, \text{Precondition}, \{\text{Service}\}, \text{SuccessRate}, \text{ExecutionTime}>\), where Postcondition is the goal wanted to be achieved. Precondition are the initial conditions that
must be given to start the execution of services necessary to fulfill the goal. Service is the list of services that must be executed from the state Precondition to reach the state Postcondition. SuccessRate indicates the percentage of executions successfully completed in the past. This term represents implicitly the confidence that the system has about this composition. Finally, ExecutionTime is the time required for the execution of the services. This value is obtained by calculating the worst-case execution times of each of the services included in the composition and combining them following the process model of the composition. This ExecutionTime term is performed to get a temporal estimation of the execution of the whole composition and to use it in temporal commitments. Temporal commitments will be shown in Section IV-B. An example of the used case-base is presented in Table I, where P is Precondition, Q is Postcondition, SR is SuccessRate and ET is ExecutionTime.

To complete the search of a service composition, the agent will inform about the activated goal (Postcondition) and its knowledge base (Precondition). With this information the On-line Planner can fulfill a service composition. To do it, the planner extracts cases from the case-base and composes a path from the goal to be achieved until it reaches any of the beliefs that are stored in the agent.

Let’s imagine the following situation using the information in Table I. An agent has as requirement the fulfillment of the goal F, and has in its knowledge base the items \{A,B\}. These items can be used as preconditions for the fulfillment of the goal. The On-line Planner will extract from the case-base all cases that have the goal F as Postcondition. For every extracted case the algorithm will come to search in the case-base, but now Postcondition is the set of preconditions of all the extracted cases (Precondition parameter). This process will follow until it extracts a case whose Precondition is either defined in the agent’s knowledge base (Precondition = A \lor B). In Figure 3 we can see the progress of the search from F to A or B. In this case, several plans are possible. In response to the needs of both the agent or the Operating System just one plan will be chosen. In order to get a result with the best success rate, any of the plans marked as (3) is picked. If it is required to get a plan that reaches the goal as soon as possible, the plan marked as (1) will be chosen.

Finally, if a plan that meets within a specified temporal bound (e.g. before 22 time units) and with the highest success rate is required, then the option (2) will be selected. As shown, the execution module can choose a plan taking into account the agent requirements, making the system more adaptable.

<table>
<thead>
<tr>
<th>Q</th>
<th>P</th>
<th>Services</th>
<th>SR</th>
<th>ET</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
<td>{S1}</td>
<td>1</td>
<td>4t</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>{S1,S3}</td>
<td>0.85</td>
<td>10t</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
<td>{S3}</td>
<td>0.85</td>
<td>6t</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>{S6}</td>
<td>0.9</td>
<td>7t</td>
</tr>
<tr>
<td>E</td>
<td>B</td>
<td>{S7,S10,S11}</td>
<td>0.76</td>
<td>11t</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
<td>{S8}</td>
<td>0.99</td>
<td>3t</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
<td>{S4,S12}</td>
<td>0.98</td>
<td>7t</td>
</tr>
<tr>
<td>F</td>
<td>C</td>
<td>{S5,S9}</td>
<td>0.81</td>
<td>7t</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>{S13,S14}</td>
<td>0.98</td>
<td>10t</td>
</tr>
</tbody>
</table>

To complete the search of a service composition, the agent will inform about the activated goal (Postcondition) and its knowledge base (Precondition). With this information the On-line Planner can fulfill a service composition. To do it, the planner extracts cases from the case-base and composes a path from the goal to be achieved until it reaches any of the beliefs that are stored in the agent.

Let’s imagine the following situation using the information in Table I. An agent has as requirement the fulfillment of the goal F, and has in its knowledge base the items \{A,B\}. These items can be used as preconditions for the fulfillment of the goal. The On-line Planner will extract from the case-base all cases that have the goal F as Postcondition. For every extracted case the algorithm will come to search in the case-base, but now Postcondition is the set of preconditions of all the extracted cases (Precondition parameter). This process will follow until it extracts a case whose Precondition is either defined in the agent’s knowledge base (Precondition = A \lor B). In Figure 3 we can see the progress of the search from F to A or B. In this case, several plans are possible. In response to the needs of both the agent or the Operating System just one plan will be chosen. In order to get a result with the best success rate, any of the plans marked as (3) is picked. If it is required to get a plan that reaches the goal as soon as possible, the plan marked as (1) will be chosen.

Finally, if a plan that meets within a specified temporal bound (e.g. before 22 time units) and with the highest success rate is required, then the option (2) will be selected. As shown, the execution module can choose a plan taking into account the agent requirements, making the system more adaptable.

This component applies a first classification to select a plan that fulfills the active goal within the agent requirements. This classification has into account past executions of the goals that have been retrieved from the TB-CBP. The plan selection is done by using the success rate and the execution time of the retrieved compositions. Thus, the On-line Planner uses static knowledge to select the plan but it has not into account the environment conditions and the agents workload at the current moment. Since a plan is composed by single services and each service can be provided by different agents at the same moment, a second classification must be done to be able to select the best providers for each single service of the plan. This classification does take into account the security mechanisms of the provider host and the agents workload at the current moment. This function is performed by the Commitment Manager which is presented below.

### B. Commitment Manager

This component is designed to select the best provider agents that offer the single services of a plan that has been selected by the On-line Planner. The Commitment Manager is related to a framework called SAES [21] which allows us to compose services and to ensure their fulfillment on time. The main difference with the SAES approach is that, by introducing the service framework as part of the operating system, it has more information to make better security commitments and temporal predictions.

The main function of the Commitment Manager is to check if the set of services offered as a plan by the On-Line Planner will be available to fulfill the request and to establish two kinds of commitments with the agents that provide the selected services: security and temporal commitments.

**Security Commitments:** One of the main purposes of this work is to establish a secure environment where services can be invoked with some grants of privacy, integrity and access control. If we focus on the current challenges in infrastructure security we can organize them in three levels: the Network level, the Host level and the Application level [22]. Since there are different network topologies, both in public and private clouds, and the Commitment Manager has no management
abilities over the network, the Network level is out of the scope of this work. As the Commitment Manager is integrated into the OS we can afford security concerns at the Host Level. At this level we can apply Host-based Intrusion Detection Systems (HIDS) to keep data integrity. At this level it is also possible to apply audit mechanisms and server virtualization. The access control mechanism can be also placed at this level. Finally, the Application level depends exclusively on the application program, this is, the agents. At this level the agent can manage encryption mechanisms, application authentication and authorization and secure coding.

The Commitment Manager classifies the security level of a service in three categories: authentication[23], detection[24] and encryption[25]. The better the security level is in each category, the more confidence will be deposited at the service provider. Each CM has a table to prioritize different mechanisms for each category: cryptographic algorithms for the encryption level, intrusion detection systems and firewalls for the detection level and access control systems for the authentication level. As an example, Table II shows a little set of authentication mechanisms and how the Commitment Manager would rate them.

<table>
<thead>
<tr>
<th>Auth mechanism</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-time password</td>
<td>0.95</td>
</tr>
<tr>
<td>Time-based authentication</td>
<td>0.87</td>
</tr>
<tr>
<td>Two factor authentication</td>
<td>0.91</td>
</tr>
<tr>
<td>Closed-loop authentication</td>
<td>0.73</td>
</tr>
<tr>
<td>Username and password</td>
<td>0.1</td>
</tr>
<tr>
<td>Digest Access authentication</td>
<td>0.4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The CM asks for security information to each agent that is providing any of the services included in the selected plan. Each agent answers notifying its security level with the tuple \(< A, D, E >\), where A stands for authentication and authorization mechanisms, D stands for detection and auditing mechanisms and finally E stands for encryption mechanisms. The CM assigns a value to the security commitments by adding the values of the tables that represent each of the A, D, E categories (like Table II) and applies a weight with the confidence that the CM has in the provider agent. If security has failed too much in the past and the agent has been penalized, its confidence will be low and, therefore, the applied weight will be low.

**Temporal Commitments:** To establish this kind of commitments the Commitment Manager sends a call for proposals to all agents that can offer the services involved in the composed service (see Figure 4). Each agent analyses when the service can completed, and then each agent returns a proposal to the Commitment Manager. The proposal consists of a tuple \(< T_{start}, T_{duration}, P >\) where \(T_{start}\) indicates the moment when the service can start its execution, \(T_{duration}\) indicates the necessary time to complete the service and \(P\) is the probability, as said by the provider agent, of finishing the service successfully. This value represents the probability of reaching a successful execution and is extracted from its success rate of executions.

On one hand, the On-line Planner obtains a quality measure (the success rate) which is used to estimate the best plan. On the other hand, the Commitment Manager calculates a probability that indicates if the composition can be completed in time, taking into account the service provider agents workload. This information refines the success rate obtained by the On-line Planner because it takes into account the current situation of the agent that offers the service and the real conditions of the environment. With all this information, a pre-commitment between the agent and the Commitment Manager is established.

When all agents have answered to the Commitment Manager, the CM must calculate the success probability associated to the whole service composition. To do that, the Commitment Manager uses the \(P\) value, which was sent by all agents. This probability is weighted with the confidence that the Commitment Manager has on these agents. The service composition success probability (SCSP) is calculated as follows:

\[
SCSP = \prod_{i=0}^{N} P_i * \omega_i
\]

where \(\omega_i \in [0, 1]\) is the weight associated to the agent that provides the service. This weight is related to the previously fulfilled commitments and represents the confidence that the Commitment Manager has on this agent; e.g. an agent that has many unfulfilled commitments will have a low confidence.

Once the Commitment Manager calculates the security level and the service composition success probability, it sends the composed service and the temporal and security commitments to the deliberation engine. The deliberation engine analyses if it is a suitable composition. If it agrees with the service composition, it communicates to the Runtime Engine that the
service executions can start. When this is the case, the pre-commitments established with the agents are confirmed by the Commitment Manager. If the deliberation engine does not agree with the service composition, the Commitment Manager breaks the pre-commitments, freeing the slack reserved by the agents.

The Commitment Manager is also in charge of ensuring that the acquired commitments are fulfilled. In case where a commitment cannot be fulfilled, the Commitment Manager penalizes the agent which provides the service. This penalty is captured through the confidence weights that are applied when the Commitment Manager updates the service composition success probability.

V. TESTS AND RESULTS

As part of this work, a simulator has been developed to demonstrate the viability of using the architecture presented here to deploy service-oriented operating systems. This software is a representative simulation environment to test different approaches of the paradigm presented in this work. The simulator allows us to change several parameters of the operating system execution without the complexity of a real operating system and with instrumented measures that do not modify the OS behavior. This simulator has shown how the deliberation engine (including mainly an On-Line Planner component based on a TB-CBP and the Commitment Manager) can adapt itself to respond to changes in the environment and different workloads and user preferences. The simulator has been designed to run a huge set of agents with different goals to achieve and with diverse services to provide.

These tests have the purpose of showing the adaptability of the system in different situations of the workload and user preferences. To simulate this, during the tests two variables have been parametrized. These variables are established by the Deliberation Engine to manage the quality and quantity of plans that are accepted to be executed.

This two variables are the maximum slack time that the deliberation engine gives to run the plan and the minimum quality that the deliberation engine requires (this quality value is the SCSP provided by the Commitment Manager). Also, the number of agents has been continuously increased at each iteration to get a bigger number of active goals, which increases the system’s workload.

The maximum slack time (MaxTime) represents the amount of time that the deliberation engine is prepared to give for the execution of a plan. Any plan whose time execution prediction exceeds this parameter will be excluded. The minimum quality accepted (MinQ) by the deliberation engine represents the lower limit that is accepted among all the plans proposed by the On-Line Planner. The quality of every plan is obtained by the Commitment Manager and represents the probability of success in the execution of the plan (SCSP).

Both parameters, the MaxTime and the MinQ, are dynamically adjusted by the deliberation engine to adapt itself to the current requirements of the system. It will always try to minimize the execution time and maximize the quality obtained. But in these tests we are going to study how both parameters modify the ratio of accepted plans, because we do not want a system that is so strict that does not accept new jobs, since its main purpose is to run plans. This is the reason why these parameters (MaxTime and MinQ) are dynamic, to adapt them to the current demanded workload of the system.

To see how the system workload affects to the number of accepted plans a stress test has been performed. The method used in this test set to stress the system has been putting a high load by increasing the number of agents. The more agents in the system, more activated goals, which results in more running plans. Figure 5 shows the variation of the number of accepted plans when the number of agents is increased, parametrizing the maximum time (MaxTime) and the minimum quality (MinQ). In this work we are showing the evolution of the accepted plans ratio when the number of agents is increased for sets of 20, 30, 50, 70, 90 and 100 agents. In these figures it is appreciated that the behavior is what is expected and desirable. This means that the accepted ratio is decreased when the workload is increased. Comparing each graph, shows a progressive decrease in the number of plans accepted when the number of agents is larger and the system is more stressed. The higher ratios occur for the worst time and quality parameters. Therefore is the responsibility of the deliberation engine to balance the parameters of time and quality to maximize the utilization of the system. Meanwhile the execution time parameter will be minimized and the quality parameter will be maximized to improve the results of each executed plan.

To analyze with more detail how these parameters modify the goodness of selected plans we have designed another test. This test (Figure 6) presents the relationship between the maximum time to achieve the plan, the minimum requested quality and how this affects to the number of plans that finish before their deadline. For lower time values the percentage of achieved plans is higher because the number of accepted plans is very low. This is because there is not enough time to execute the most of the compiled plans. We can also see that for high values of quality the number of plans that finish in time is always high. Therefore, it is observed the correct behavior of the deliberation engine, which is able to predict the execution of the services to fulfill the required quality parameters and select only the plans that will fulfill their commitments.

Finally, an specific test has been done to check the relevance of the security commitments. This test is intended to check how important is selecting a provider that applies the proper security policies to transactions. It is also very important to trust your provider, so your providers confidence level must be dynamic and well calculated. For this test we have prepared a scenario where a client agent activates a goal that is fulfilled by a set of agents, each one of them with a different security level. This test is run with three different OS configurations: an OS without Commitment Manager that does not apply any kind of commitment when selecting a service (the selection is random); an OS with a CM that only performs temporal commitments (t-commitments), however, no security commitments are done. That is, it selects the service with a lower execution time and a higher success rate; Finally, a third OS with a CM that performs t-commitments and s-commitments
In this case security and temporal parameters are taken into account. To test the robustness of the system we have introduced a sniffer in the simulator that spies the communications between all the agents, trying to steal passwords that are exchanged when invoking services. If the security level of the service is high, the password is encrypted with a strong algorithm. If the security level is very high the access control uses one-time passwords, this makes very difficult to steal a useful password. When the security level is low the password is very easy to be decrypted. Finally, when the security level is very low the password is not encrypted. As shown in Figure 7, the percentage of stolen passwords for a system with s-commitments is very low. As long as the services that are selected for execution are those that have the highest security levels, passwords are rarely stolen. It improves with time as the confidence is better calculated, based on experience. Nevertheless, the percentage of stolen passwords is very similar when the CM performs only t-commitments and when there is no CM in the OS. This is because the selection of services based on security parameters is completely random. We can even see that, in some cases, the percentage of stolen passwords is higher with t-commitments than without commitments. This is because security algorithms are usually big time consumers, so the selected services tend to be those with the lowest security level.

VI. CONCLUSIONS

In this work it has been presented a deliberation engine for a service-oriented operating system based on the agent
technology. This OS uses a goal-oriented computing paradigm where the entities of the system express their goals. The OS is in charge of helping the achieving of these goals by means of a service-oriented approach. This feature is provided by a deliberation engine that is capable of obtaining the services needed to achieve the users goals, taking into account time and security constraints. This is done either to select the providers with the highest security level, to obtain a result before a deadline, or simply to improve the success probability of the result.

The deliberation engine takes into account security, temporal and quality parameters in order to obtain executions with a high success degree. To do it, a Commitment Manager has been introduced into the deliberation engine of the OS to establish temporal commitments and security commitments. It has also been introduced a Temporal Bounded CBP to compose on-line plans. This On-Line Planner has allowed us to compose plans that give solutions to the goals of the agents following temporal constraints. To guarantee that the agents execute their services before their deadline, the Commitment Manager is in charge of analyzing the workload and establishing a t-commitment between the agents and the deliberation engine. The CM also analyzes the security mechanisms provided by the services and the confidence that it has in each service and establishes a s-commitment to ensure privacy, integrity and access control in transactions.

The results of this work have shown how the deliberation engine can modify time and quality parameters to adjust the workload of the system using the experience acquired in its own case-base. We have shown also how the OS that makes s-commitments tends to use the more dependable providers, resulting in a decreasing number of attacks carried out successfully. This way, the OS becomes an adaptive system capable of deciding which plans can be executed to improve the quality of the results within a secure environment.

This proposal opens the possibility of designing service-oriented operating systems directed by goals. These new kind of OS can be extended continuously with new services and plans driven by the user needs. These plans are added by means of the services offered by other users and by their composition thanks to the new goals defined by the users.

The OS architecture defined in this work allows us to use this goal-oriented computing paradigm, since there are some capabilities that only the OS can provide (like soft real-time constraints, security commitments and temporal commitments).

ACKNOWLEDGMENTS

This work is supported by TIN2008-04446 and TIN2009-13839-C03-01 projects of the Spanish government, PROMETEO/2008/051 project, FEDER funds and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

REFERENCES


