Run-time Safety Framework for Component-based Medical Robots

Min Yang Jung
Department of Computer Science
Johns Hopkins University
Baltimore, Maryland 21218
myj@jhu.edu

Peter Kazanzides
Department of Computer Science
Johns Hopkins University
Baltimore, Maryland 21218
pkaz@jhu.edu

ABSTRACT

As modern medical robot systems are required to perform complex surgical tasks with various sensing and actuation capabilities, it is becoming more important to integrate a variety of sensors, actuators, and control loops into a single system. However, the consideration of nonfunctional properties such as performance and fault tolerance complicates the integration and makes it harder to achieve system safety. In medical robotics, such system issues have not received much attention despite a consensus on the importance of safety within the domain. As our approach to this issue, we present a run-time software environment for safety research on component-based medical robot systems, called the Safety Framework. This framework aims to provide systematic safety methods by decomposing safety features into reusable safety mechanisms and safety specifications. This decomposition enables the accumulation of safety experience and knowledge in a traceable manner, and provides reusable safety design guidelines for designing new medical robot systems.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*software libraries*; D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures, patterns*; D.2.13 [**Software Engineering**]: Reusable Software—*domain engineering, reusable libraries, reuse models*

General Terms

Design

Keywords

Medical robotics, software system safety, component-based software engineering, system architecture, layered architecture

1. INTRODUCTION

Medical and surgical robots are examples of safety-critical systems due to the potential hazards that can lead to severe injury or the loss of human life. Since an industrial robot was first used for stereotactic neurosugery in 1985 [38], a variety of medical and surgical robot systems has been developed in diverse application areas to enable less invasive and more accurate surgeries. Commercial robot systems are also actively used in modern operating rooms.

As modern medical robot systems are required to perform different surgical tasks with various sensing and actuation capabilities, the complexity and scale of systems have been rapidly increasing to meet both functional requirements (e.g., real-time sensing and actuation, various types of sensory feedback, control of robots at high frequency) and nonfunctional requirements (e.g., safety, performance, reliability). In such large scale and highly complex systems, it becomes more complicated to achieve system safety. In robotics, these system issues have already appeared and component-based software engineering (CBSE) [39] has been widely adopted [35, 3] as an effective solution. The robotics domain has benefited from many open-source robotics software packages (e.g., Orocos, ROS, OpenRTM, CLARAty, Player, OPRoS), mainly for research on the *functional* aspects of robot systems.

This trend applies to the medical robotics domain as well and the nonfunctional properties–especially safety–have not yet received much attention. However, when designing safety-critical systems, it is a well known fact that safety has to be considered up front throughout the entire development process because it is easier and cheaper to build safety in than to add it later [33]. Despite an agreement on the importance of safety, systematic methods to build medical robot systems with consideration of safety have not yet been actively investigated. As an approach to this, we propose the Safety Framework (SF), a run-time software platform for component-based medical and surgical robot systems. The idea is to decompose safety features or implementation into *reusable safety mechanisms* and *safety specifications*. Our assumption is that there is enough in common between most medical robotic systems and thus it is possible to extract reusable safety mechanisms, and we confirm this assumption through literature review in the medical robotics domain.

Another key concept of SF is to adopt established methods and foundations from relevant domains such as software engineering, system safety engineering, and dependable computing. The knowledge and experience of those domains could provide insight and guidelines for our approach. For this, we look at relevant domains focusing on safety (Sec. 2)

and adopt dependability formalism and semantics as the foundations of SF. We consider domain-specific characteristics of medical robotics in Sec. 3 and describe the architecture of SF in Sec. 4. To facilitate the system development process, SF provides system designers a set of useful elements and tools that allow reusable safety mechanisms to be easily instantiated, systematically configured, and flexibly deployed to the system. In Sec. 5, we present a classification of safety design methods that have been frequently used in the medical robotics domain based on our literature review, and show how these safety design methods can be realized within SF.

This work aims to design a domain-specific research platform where safety mechanisms for component-based medical robots can be systematically analyzed and investigated. In the longer term, we hope to provide reusable safety mechanisms to: (1) facilitate the development of safe medical robot systems, (2) enable the accumulation of safety knowledge and experience, (3) replace application-specific safety solutions by systematic solutions, and (4) facilitate the testing and certification of these systems.

2. RELATED WORKS

Safety has been extensively investigated in a variety of domains and disciplines, both in academia and in industry, where systems are required to be safety-critical. Relevant disciplines include dependable computing, control systems, software engineering, systems engineering, and safety engineering. Examples of safety-critical systems are automotive systems, aerospace systems, medical devices, nuclear power plants, chemical processing plants, airflight control systems, railway control systems, and weapon control systems.

Safety is one attribute of dependability, for which concepts and foundations are established in the dependable computing domain [1]. In the context of dependability, faults, fault detection and diagnosis, and fault management are of major concern and these topics have been also studied in other disciplines such as chemical engineering [48] and control systems [51].

In CBSE, one recent trend is to integrate traditional safety analysis techniques and methods with a component model. Kaiser et al. [23] proposed an idea that embeds traditional fault tree analysis (FTA) into components, Component Fault Tree, which allows partition of fault trees into multiple components. Grunske integrated component-based safety evaluation techniques and failure propagation models with the SaveCCM component model for the early estimation of failure and hazard probabilities [15]. Another approach to integrate safety analysis into component-based software systems is the Safe Component Model (SCM) by Domis and Trapp [10]. Based on the concept of separation of concerns, a SCM component defines the specification layer and realization layer and each layer requires both functional and failure information. In this way, safety specification of components can be derived and analyzed independently from functional specifications.

One motivational concept in the system safety domain is the safety kernel [50] or safety executive [32] where safety issues are brought to the forefront by separating safety mechanism from policy and the concept of reuse of safety was introduced. Leveson proposed systems approaches for safety (Safeware [33] and STAMP [31]). Another interesting work is the safety case pattern [30] that proposes an approach to the development, presentation, maintenance, and reuse of safety cases based on the goal structuring notation.

Safety in robotics has been studied for fault detection, fault tolerance, and human-robot interaction. Visinsky et al. [49] introduced an approach to integrate fault detection and fault tolerance within an expert system framework. Haddadin [18, 17] investigated safety issues in human-robot interaction focusing on collision avoidance and the analysis of injury characteristics due to moving robots.

In medical robotics, several commercial surgical robot systems and a wide variety of research-level systems have been developed. The consideration of safety and the adoption of software engineering techniques started to appear in the literature around the early 1990's. Kazanzides et al. applied the concept of states and a state machine to the development of a commercial orthopaedic surgery robot system [29]. Varley [47] described a set of techniques used in practice (e.g., IEC1508) to develop a commercial endoscopic camera manipulator approved for clinical use. Rovetta [37] described a list of safety features of a telesurgery system for telerobotic prostate biopsy on human patients, based on the European Union directives with technical standards such as IEC 204 or ISO 10218. Cleary et al. [5, 14] adopted the state machine as the fundamental mechanism for inherent safety, determinism, repeatability, and testability of the Image Guided Surgery Toolkit (IGSTK), a component-based software system for image-guided surgery applications. Kazanzides [25] presents a tutorial overview of safety design for medical robots and discusses safety considerations such as safety requirements, risk assessment, and safety design. More recently, Muradore et al. [34] applied formal methods to the verification of safety properties of a surgical tool and demonstrated its feasibility using a puncturing task. Kazanzides et al. [27] applied formal methods to prove the correctness of the concurrent data exchange mechanisms of an open source component-based software package.

Other major efforts on system safety include a set of industrial standards, articles, and guidelines from industrial robotics (e.g., ANSI R15.06-1999), the medical device industry (e.g., IEC 60601, 60812, 61508, 62304), the National Aeronautics and Space Administration (e.g., NASA-STD-8719.13B), the United States Department of Defense (e.g., MIL-STD-882D), and the U.S. Food and Drug Administration (FDA) (e.g., 510k guideline).

3. DESIGN REQUIREMENTS

From our literature review on the approaches to system safety in various domains, including medical robotics, we were convinced that more *systematic* approaches to the safety of medical robots are necessary. As a starting point for this approach, we aim to design a research platform where safety mechanisms for component-based medical robots can be systematically analyzed and investigated. When designing such a platform, a large body of work from other safety-critical systems could provide insight and guidance based on their domain knowledge and experience. However, they may not be directly applicable to the medical robotics domain due to domain-specific characteristics that differentiate it from the other domains. This section describes the design requirements considering the domain characteristics of medical robotics.

3.1 Monitoring and Event Handling

The workspace of medical robot systems is mostly *un-*

structured because it operates on or inside a patient's body, which is in constant movement due to heart beats, respiration, or other physiological processes. These movements lead to *unpredictability* of the surrounding environment from the robot's perspective. Furthermore, each patient has different anatomical structures that vary in size and shape (*variability*). Traditional surgeries relied on the surgeons' expertise (i.e., knowledge, experience, skill) to deal with these issues, whereas robotic surgery exploits various types of information such as patient-specific pre-operative images (e.g., CT or MRI), real-time stereo HD video, and force feedback. For this reason, medical robot systems are information-intensive systems and can be called *Computer-Integrated Surgery* (CIS) systems [46, 40]. In such complex systems, the robot system may encounter hardware failures or high-level software faults. It is often sufficient to make a system *fail-safe* in these cases [25] and two essential mechanisms to achieve this requirement are: (1) monitoring of the surrounding environment, and (2) initiation of reflex action. Thus, a safety research platform is required to provide *monitoring mechanisms* and *event handling mechanisms* for reflex actions.

3.2 Human Factors

In medical robotics, one widely accepted control scheme is *human-in-the-loop* control where the surgeon actively participates in the control loop in order to have full control over the system during the entire procedure. Thus, the surgeon relies on the indirect sensory feedback information that the robot system provides. In such human/machine cooperative systems, human-computer interface issues are likely to arise and thus have to be considered. It is also important to help the surgeon be able to make informed decisions during the procedure. The online visualization of the procedural progress with the current state of the system is one effective method that has been frequently used in the domain. To facilitate this visual feedback, the monitoring subsystem is required to be designed such that monitor instances can be easily configured to use the visualization framework in order to stream and visualize internal states of the system.

3.3 Reusability, Testability, Traceability

Medical robot systems have been developed for a broad range of inherently different applications, such as stereotactic brain surgery, orthopaedics, microsurgery, and otolaryngology. Each surgical application has highly procedure-specific requirements and thus safety mechanisms have been developed in an application-specific manner. As an approach to this issue of application diversity, services and mechanisms that a safety research platform provides have to be *reusable*.

Another issue is that medical robot systems must be approved by regulatory agencies such as FDA to be used for clinical tests or released as commercial products. This approval process requires extensive documentation that shows the design history and traceability between requirements, implementation, testing, verification and validation. However, specific design or manufacturing procedures are not, in general, dictated by the medical device regulatory agencies and thus it is possible to use a framework to assist with the development process. To facilitate this development process, the framework design has to support systematic *testing* with *traceability*. If the framework supports reusable safety mechanisms, whose behavior can be easily specified by configuration parameters, tests of the safety features can be performed systematically and extensively by only changing configuration parameters, and the safety of the overall system is then determined by a set of safety configuration parameters.

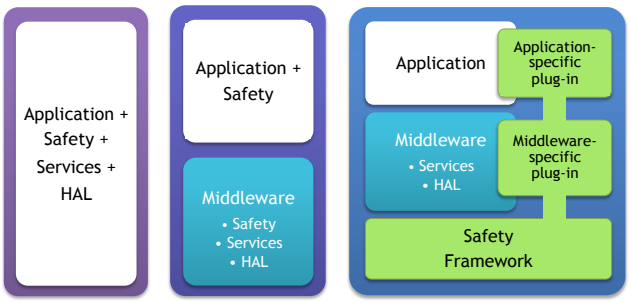


Figure 1: Evolution of robot system architectures: (a) monolithic architecture (HAL: hardware abstraction layer), (b) middleware-based architecture, (c) Safety Framework-based architecture

4. SAFETY FRAMEWORK

We present the Safety Framework (SF), which aims to establish a run-time software framework for systematic safety research in component-based medical robotics. The goal of SF is to facilitate the development process of component-based medical robot systems by providing a set of reusable, configurable, and essential mechanisms and services for safety. These mechanisms and services are designed to be middleware- and application-independent so that various component-based robotics middlewares and frameworks [1] can be used with SF to build systems for diverse applications in medical robotics. For system analysis and safety requirement definitions, SF relies on established conventional methods such as FTA and failure mode, effects, and criticality analysis (FMECA).

SF is being developed as an open source C++ framework. It consists of a collection of libraries, executable examples, tool support for visualization and a lightweight database. We currently use the *cisst* package [24] as an underlying component-based middleware, although the middleware independence of SF allows us to easily replace *cisst* with other component-based frameworks.

4.1 Design Concept and Architecture

Software system architectures for robot applications have been evolving over time. In the early days, the entire system was a single entity (*monolithic architecture*, Fig. 1a) that had all system elements such as application logic, safety features, common services (e.g., logging), and hardware abstraction layer (HAL). This architecture turned out to be problematic as the scale and complexity of the system increased due to, for example, high maintenance costs and low reusability. To deal with these issues, the concept of *separation and reuse* was introduced to split a system into two layers, the Application layer and the Middleware layer. By middleware, we refer both to the network communication infrastructure between components and to the basic robotic functionality

[1] Elkady *et al.* [12] presents a comprehensive survey on component-based middlewares and packages for robotics.

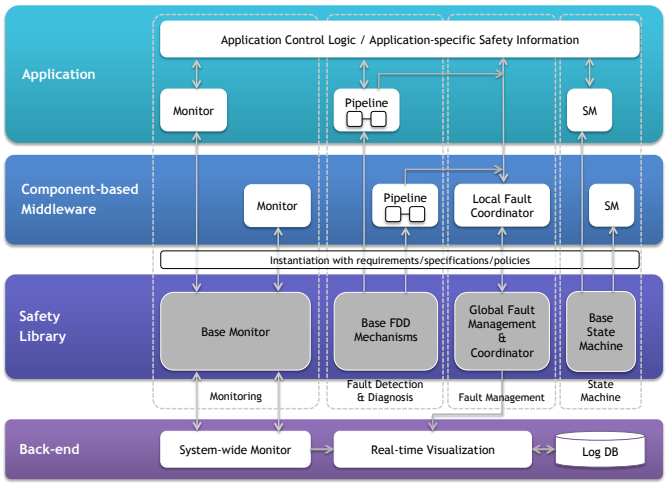


Figure 2: The architecture of a system with Safety Framework. Four essential mechanisms for safety are represented as grey boxes and are wrapped with vertical dashed boxes.

(e.g., servo control, kinematics, trajectory generation) that is relevant to all applications. This leads to the *middleware-based architecture* shown in Fig. 1b. This separation enabled reuse of middlewares for different applications and allowed applications to focus more on application-specific logic and safety features. Likewise, we apply the concept of separation and reuse to the middleware-based architecture to extract safety mechanisms from the system, and introduce a SF that provides essential services for reusable safety mechanisms. We call this an *SF-based architecture* (Fig. 1c). In this architecture, application- and middleware-specific plug-ins enable data exchange between the Application/Middleware layers and SF, while allowing SF to have control over the both layers.

4.1.1 Layered Architecture

As in Fig. 1, SF-based systems consist of the Application layer, the Middleware layer, and SF. SF is comprised of two layers: (1) the *Safety Library* layer to provide essential mechanisms for safety, and (2) the *Back-end* layer to support additional features such as visualization or database integration. Fig. 2 shows the overall architecture of SF.

The *Safety Library* layer is the core part of SF and provides four services that are essential for reusable safety mechanisms: *monitoring*, *fault detection and diagnosis (FDD)*, *fault management*, and *state machine* (Sec. 4.2 describes these mechanisms in more detail). The Middleware and Application layer instantiate these mechanisms with middleware- or application-specific requirements. The idea is to enable flexible configuration and easy deployment of the services and safety mechanisms that SF provides.

The *Back-end* layer consists of a set of additional tools and services for data visualization and database support. SF uses the Data-Driven Documents (D3) library [2] as a web-based visualization library and provides auxiliary tools to visualize run-time information of the system in intuitive and effective ways. In complex and information-intensive systems such as CIS systems, raw numbers or simple bar and line graphs are often not sufficient to monitor the current status or performance of the system. We use this visualization back-end as a basis for investigating human-computer interface issues in the safety design. Another back-end service that SF provides is database support that can be used to archive data of the system. This feature can be particulary useful when data generated during the surgery needs to be archived for future reference or post analysis purposes. This database back-end is also useful for visualization purposes in that the effective visualization of run-time data sometimes requires the history of data for data aggregation or statistical operations on the collected data. Currently, SF supports the MongoDB [4], a lightweight, scalable, and open source NoSQL database.

4.1.2 Deployment Architecture

SF is a *run-time* environment for component-based systems and aims to allow system designers to be able to easily instantiate, deploy, and test safety mechanisms with minimal application- or middleware-level code changes. Our approach to achieve this is the use of a plug-in architecture that hides SF-specific APIs as much as possible, while providing concise and well-defined APIs for the Middleware and the Application layers.

A typical component-based multi-process system is shown in Fig. 3, where two high-level applications exchange data with each other. When the SF is integrated with this system, the system is *augmented* with SF as in Fig. 4. This integration process begins with the creation of a *SF instance* which internally creates the *Safety Coordinator (SC)* as a singleton object in each user process, and then instantiates plug-ins for the Application and Middleware layers. SC–the "local fault coordinator" in Fig. 2–has full access to all information in the same process, and can also coordinate all components in the same process, via plug-ins. The *Safety Supervisor (SS)*–the "global fault management and coordinator" in Fig. 2–is unique within the system and primarily manages *"global"* knowledge of the system, whereas SC focuses on *"local"* knowledge within its process boundary. The idea behind this design is to build a hierarchy in knowledge coverage and access for effective fault management and coordination. For example, some faults can be removed locally by SC, i.e., without the intervention of SS, whereas the coordination of SS and SCs may be required for some other faults.

For data exchange between SCs and SS, SF uses an independent network communication infrastructure. This increases the redundancy of message exchange channels and the reusability of SF for different middleware packages. Currently, SF uses IceStorm [19], a publish and subscribe event distribution service from ZeroC, but it can be replaced with other network infrasturcture because the interface between SF and the network infrastructure is designed based on the Abstract Factory Pattern [13] to increase reusability.

With SF in place, the previous system in Fig. 3 becomes Fig. 5. The only "visible" modifications to the original system

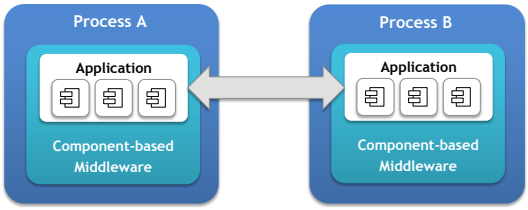


Figure 3: Multi-process system with component-based middleware

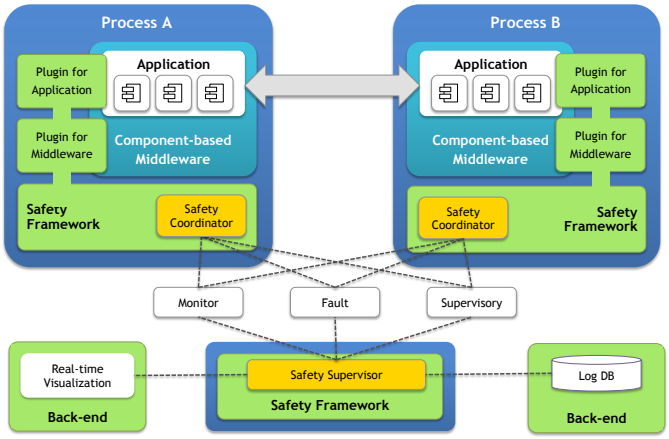


Figure 4: A system integrated with the Safety Framework (SF colored in green)

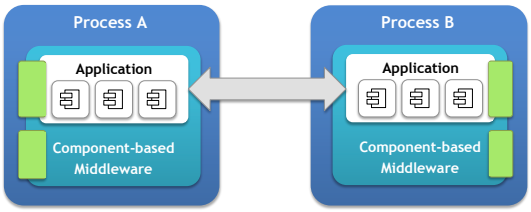


Figure 5: Deployment view of the original system with SF. SF-specific parts are in green.

are merely specifications to instantiate safety mechanisms or services that SF provides.

4.2 Services for Safety

In medical robotics, a *fail-safe* system is often sufficient because the robots can be powered off to be brought to a safe state [25]. However, more *fault-tolerant* systems with complex computation or intelligence may be required to enable more advanced and complicated surgeries, especially those surgeries which could not be completed without robotic assistance.

Fault tolerance has been extensively investigated in other areas, such as dependable computing and fault-tolerant control systems. Our literature review on those domains[2] identified three essential mechanisms for fault-tolerance: *monitoring*, *fault detection and diagnosis*, and *fault removal*.

Currently, SF supports the first two mechanisms and these are described in the following sections. The support for fault removal is a work-in-progress and will be added later in the context of fault management and coordination. Other mechanisms under investigation (and not discussed further in this paper) are the state machine mechanism that allows system designers to handle system states in more systematic and structured manners, and the effective visualization methods that improve the presentation of the current system or application status using the SF visualization back-end.

4.2.1 Monitoring

[2]A few samples of motivational models found in the literature include the Integration Framework [36], the general scheme of supervision, fault detection, and fault diagnosis methods [20], and the general structure of active fault-tolerant control systems [51].

Listing 1: Example of how to instantiate `cisstMonitor` to monitor the actual thread execution period of a robot's force sensor interface component

```
1  // define target
2  cisstLocation * target = new cisstLocation;
3  target->SetProcessName("Robot");
4  target->SetComponentName("ForceSensor");
5  // create monitor
6  cisstMonitor * monitor =
7    new cisstMonitor(
8      SF::Monitor::THREAD_PERIOD, // actual thread period
9      target, // target information
10     SF::Monitor::ON, // enable monitor at start up
11     SF::Monitor::STREAM, // periodic sampling
12     100); // sampling rate (Hz)
13 SafetyCoordinator()->AddMonitor(monitor);
```

In medical robotics, system monitoring has been widely used since the late 1980's [8, 43, 11, 45] and has been recognized as an essential tool to improve system safety. Because of its effectiveness and wide acceptance, SF supports the monitoring mechanism as one of the essential services for safety. The SF monitoring mechanism is designed with the following objectives:

- *Minimal run-time overhead*: Monitors are required to have minimal run-time overhead on the system.
- *Flexible configuration*: A monitoring behavior needs to be flexibly configured.
- *Convenient deployment*: A monitor should be easily deployed to the system.

Because SF has no direct access to the internal data of the Middleware or Application layer, SF relies on the plug-ins to fetch the internal data. Currently, SF uses *cisst* as a middleware, which provides thread-safe efficient data exchange mechanisms between components [26], and thus SF can sample data with minimal run-time overhead. The SF APIs are designed to provide a set of configuration parameters to specify what to monitor (e.g., thread scheduling latency, thread over-run), how to monitor, and the sampling rate for monitoring. This allows monitor instances to be flexibly configured and instantiated. To deploy the instantiated monitors, SF communicates with the middleware to coordinate the monitor deployment process via the SF plug-in for the middleware.

Listing 1 shows an example of how to instantiate a monitor using a set of configuration parameters. In this example, we first specify the location of a component to monitor (lines 2-4 and 9) which, according to the *cisst* component model [21], can be uniquely identified by a process name ("`Robot`") and a component name ("`ForceSensor`"). We want to monitor the actual thread period of the component (line 8) using periodic sampling (line 11) where data is sampled at 100 Hz (line 12). The instantiated monitor can start as soon as it gets deployed to the system (line 10) and actual deployment occurs by calling the SF API (line 13).

4.2.2 Fault Detection and Diagnosis

The starting point for fault tolerance is fault detection and diagnosis (FDD). Over the last three decades, FDD has been extensively studied in various domains such as chemical engineering, control engineering, and reliability engineering, and a wide variety of different FDD methods and techniques have been developed [51]. Although the basic concept of these methods–detecting a fault in the system and initiating

Listing 2: Example of how to instantiate a thresholding filter

```
1  SF::FilterThreshold * filter =
2    new FilterThreshold(
3      // Common arguments
4      "ForceSensor", // name of target component
5      SF::FilterBase::INTERNAL, // filtering type
6      // Filter-specific arguments
7      "Force", // name of input signal
8      0.5, // threshold (0.5 kgf)
9      0.05, // margin
10     0.0, // output for normal cases
11     1.0); // output if threshold exceeded
12 // install filter
13 SafetyCoordinator()->AddFilter(filter);
```

immediate reflex actions–is adequate for medical robotics, domain-specific characteristics should also be considered.

Considering the domain characteristics, we first describe the fault semantics of SF, which is a reduced version of the *combined fault classes* [1] from the dependable computing domain and does not currently[3] consider the following groups of faults:

- *Objective faults (malicious faults)*: SF does not consider security issues, although security could be an important topic depending on the application (e.g., teleoperation, telesurgery).
- *Intent faults (deliberate faults)*: SF assumes that the system designers or operators do not make intended bad decisions that are wrong and cause faults.
- *Capability faults (incompetence faults)*: SF users are assumed to have enough professional experience and competence not to make bad decisions that have consequences which result in mishap.

Thus, all faults of SF are assumed to be *nonmalicious*, *nondeliberate*, and *accidental*.

The FDD mechanism of SF is based on our previous work on FDD methods for component-based robotic systems [22] which include the filter, the history buffer, the filtering pipeline, and the FDD pipelines. The idea is to provide generic, customizable, and extensible "building blocks" for FDD so that middleware- or application-specific FDD methods can be easily designed, implemented, and deployed to the system. SF provides two options to deploy filters and filter pipelines (henceforth, simply "filters"): *external filtering* and *internal filtering*. The main difference between the two execution modes is which thread performs the actual computation for the filters and filter pipelines. In internal filtering, filters are processed by the thread of the component that provides the inputs to the filters, whereas external filtering deploys filters in a separate component dedicated for monitoring and FDD computation, which fetches data and runs filters. This difference leads to inherently different characteristics of filters and filtering process in terms of run-time overhead, processing delay, and guarantee of execution. It also allows system designers to have flexibility in deploying filters to the system considering application-specific characteristics and requirements.

Listing 2 illustrates how to instantiate a thresholding filter to monitor excessive forces. A real-world use case considered here is an orthopaedic surgery robot which reported that the cutting forces exerted on cadaver dog bones seldom exceed 0.5 kgf, with local maxima about 0.3-0.4 kgf [44]. The first two parameters (lines 4, 5) are required for all SF filters and specify a component to deploy the filter by its name as well as the execution mode (either internal or external). As in line 5, the two execution modes can be switched with just a single parameter change. The other parameters are filter-specific (lines 7-11) and they include a name of the input signal to the filter (line 7), a threshold (line 8), a 10% margin that could be useful to suppress false positives due to measurement errors or noise (line 9), an output value for normal cases (line 10), and another output value when the measurement exceeds the threshold (line 11).

[3]Fault classes excluded here are expected to be topics for future research.

5. SAFETY DESIGN METHODS

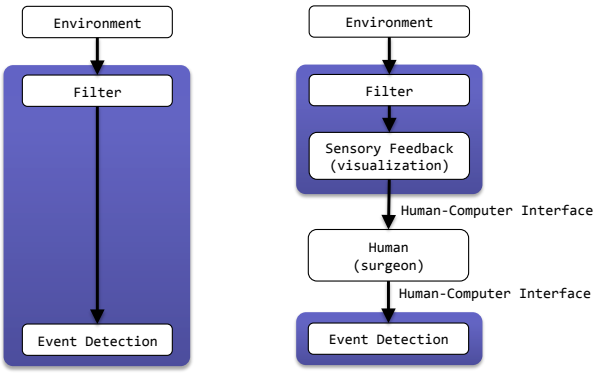
SF provides two reusable resources: the essential services for safety and the safety design mechanisms. Based on a literature review of relevant domains, we identified three essential services in Sec. 4.2. In the case of safety design mechanisms for medical robot systems, however, there is not yet an accepted safety standard for medical robot systems [25], nor a consensus of the definition of safety. Thus, we rely on published articles to understand how previous and existing medical robot systems have dealt with safety.

Based on an academic literature review of medical robot systems, starting from 1985 [38], with emphasis on the safety designs and techniques, we were able to identify a set of safety designs that have been frequently used for, and proven to be effective for, various medical robot application systems. We classified these safety designs into three categories, although more extensive review is expected to improve and elaborate this classification: (1) designs that involve hardware, (2) software-based designs, and (3) human-computer interface. Table 1 shows a list of the safety designs classified (a very short list of works referenced here includes [38, 45, 28, 41, 7, 6, 9, 42, 16]).

Class	Safety Design Methods
I	Low motor speed (slow motion) Low motor torque Fail-safe e-pause and e-stop Separate monitoring subsystem Electric circuits for power enable interlocks Use of force (and torque) sensor (force control) Redundant sensors (dual encoders) Active brake Use of accelerometer Dead-man switch
II	Redundant consistency checking Dynamic constraints (safety volume, virtual fixture) Online monitoring Safety timeout (watchdog)
III	Surgeon-in-the-loop control (intraoperative control) Visualization (visual display of current status) Devices for surgeon (control pad, pedal, pendant) Sensory feedback (audio, visual, tactile feedback)

Table 1: Classes of safety designs

These safety design methods can be considered as a process to detect events of the system in order to initiate appropriate reflex actions. Using SF services, we can model this event detection process in two different ways as in Fig. 6, depending on whether the human is involved in the process or not. The first model (Fig. 6a) involves no human intervention and is typically used for pre-programmed reflex actions based on pre-defined scenarios. This model can detect events and initiate reflex actions with minimal latency, but its reliance



(a) No human (b) Human as decision maker

Figure 6: Two models of event detection process (colored parts represent SF)

on pre-structured information requires thorough analysis of possible scenarios. One good example of this model is the monitoring of force sensor feedback with a thresholding filter, as discussed in Sec. 4.2.2. In contrast to the first model, the second model (Fig. 6b) allows the humans (e.g., surgeons, system developers, medical personnel) to actively participate in the event detection process as decision makers, based on their domain knowledge and experience. For example, the monitor instance in Listing 1 can be coupled with the SF visualization back-end to provide a surgeon with visual cues of the current force and the surgeon can hit a fail-safe emergency-stop button if the force reading from the monitor is too high.

6. CONCLUSIONS

Medical robot systems are safety-critical systems that have unique domain characteristics in operational environments, control methods, diversity of applications, and regulatory requirements. Considering these characteristics, we designed the Safety Framework (SF), a middleware-independent and application-independent safety research platform for medical robotics. The design concept of SF reflects the knowledge and experience of related domains, such as fault tolerant control systems and safety engineering, and adopted the established dependability formalism and semantics from the dependable computing domain considering domain-specific characteristics. Based on our literature review, we also summarized a list of safety design techniques that have been frequently used and proven to be effective in history, and presented a preliminary classification of safety design methods.

We continue to improve the essential services of SF, especially the state machine and fault management. This will allow us to handle faults more actively and more systematically. We will also further develop the classification of safety design techniques based on more extensive literature review in the domain. This classification will help us to identify reusable safety design mechanisms. For validation, we will apply the developed methods to actual medical robot systems (e.g., the ROBODOC® surgical robot).

Our approach to safety, SF, aims to bring the safety issue of medical robot systems up front, in order to enable early and systematic integration of safety considerations into the system development process. In this way, our goal is to establish a run-time base framework where the safety of component-based medical and surgical robotics can be systematically investigated.

7. REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. on Dependable and Secure Computing*, 1:11–33, Jan. 2004.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. on Visualization & Comp. Graphics*, 2011.

[3] D. Brugali and P. Scandurra. Component-Based Robotic Engineering (Part I). *IEEE Robotics & Automation Magazine*, 16(4):84–96, Dec. 2009.

[4] K. Chodorow and M. Dirolf. *MongoDB: the definitive guide*. O'Reilly Media, 2010.

[5] K. Cleary, L. Ibanez, S. Ranjan, and B. Blake. IGSTK: a software toolkit for image-guided surgery applications. *International Congress Series*, 1268(0):473 – 479, 2004.

[6] B. Davies, K. Fan, R. Hibberd, M. Jakopec, and S. Harris. Acrobot - using robots and surgeons synergistically in knee surgery. In *Proc. of the Int. Conf. on Advanced Robotics (ICAR)*, pages 173–178, Jul. 1997.

[7] B. L. Davies. *A discussion of safety issues for medical robots*, pages 287–298. Computer-Integrated Surgery. MIT Press, Cambridge, MA, 1996.

[8] B. L. Davies, R. D. Hibberd, M. J. Coptcoat, and J. E. A. Wickham. A surgeon robot prostatectomy - a laboratory evaluation. *Journal of Medical Engineering & Technology*, 13(6):273–277, 1989.

[9] E. Degoulange, L. Urbain, P. Caron, S. Boudet, J. Gariepy, J.-L. Megnien, F. Pierrot, and E. Dombre. HIPPOCRATE: an intrinsically safe robot for medical applications. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 2, pages 959–964, Oct. 1998.

[10] D. Domis and M. Trapp. Integrating safety analyses and component-based design. In M. Harrison and M.-A. Sujan, editors, *Computer Safety, Reliability, and Security*, volume 5219 of *Lecture Notes in Computer Science*, pages 58–71. Springer Berlin / Heidelberg, 2008.

[11] J. Drake, M. Joy, A. Goldenberg, D. Kreindler, et al. Computer-and robot-assisted resection of thalamic astrocytomas in children. *Neurosurgery*, 29(1):27–33, 1991.

[12] A. Elkady and T. Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In *European Conf. on Object-Oriented Programming (ECOOP)*, volume 707 of *Lecture Notes in Computer Science*, pages 406–431. Springer Berlin Heidelberg, 1993.

[14] K. Gary, L. Ibanez, S. Aylward, D. Gobbi, M. Blake, and K. Cleary. IGSTK: an open source software toolkit for image-guided surgery. *Computer*, 39(4):46–53, april 2006.

[15] L. Grunske. Towards an integration of standard component-based safety evaluation techniques with saveccm. In *Quality of Software Architectures*, pages 199–213. Springer, 2006.

[16] G. Guthart and J. Salisbury, J.K. The Intuitive Telesurgery System: Overview and Application. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 618–621, 2000.

[17] S. Haddadin, A. Albu-Schaffer, F. Haddadin, J. Rosmann, and G. Hirzinger. Study on soft-tissue injury in robotics. *Robotics Automation Magazine, IEEE*, 18(4):20–34, Dec. 2011.

[18] S. Haddadin, A. Albu-Schäffer, and G. Hirzinger. Requirements for safe robots: Measurements, analysis and new insights. *Intl. Journal of Robotics Research*, 28(11–12):1507–1527, 2009.

[19] M. Henning. A new approach to object-oriented middleware.

IEEE Internet Computing, 8(1):66–75, Jan-Feb 2004.

[20] R. Isermann. Supervision, fault-detection and fault-diagnosis methods–an introduction. *Control engineering practice*, 5(5):639–652, 1997.

[21] M. Y. Jung. A layered approach for identifying systematic faults of component-based software systems. In *Proc. of the 16th Intl. Workshop on Component-Oriented Programming*, pages 17–24, New York, NY, USA, 2011. ACM.

[22] M. Y. Jung and P. Kazanzides. Fault detection and diagnosis for component-based robotic systems. In *IEEE Intl. Conf. on Technologies for Practical Robot Applications (TePRA)*, Woburn, MA, USA, Apr. 2012. IEEE.

[23] B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *Proc. of the 8th Australian workshop on Safety critical systems and software*, volume 33 of *SCS '03*, pages 37–46. Australian Computer Society, Inc., 2003.

[24] A. Kapoor, A. Deguet, and P. Kazanzides. Software components and frameworks for medical robot control. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3813–3818, May 2006.

[25] P. Kazanzides. Safety design for medical robots. In *IEEE Intl. Conf. on Engineering in Medicine and Biology Society (EMBS)*, pages 7208–7211, Sep. 2009.

[26] P. Kazanzides, A. Deguet, and A. Kapoor. An architecture for safe and efficient multi-threaded robot software. In *IEEE Intl. Conf. on Technologies for Practical Robot Applications (TePRA)*, pages 89–93. IEEE, Nov. 2008.

[27] P. Kazanzides, Y. Kouskoulas, A. Deguet, and Z. Shao. Proving the correctness of concurrent robot software. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4718–4723, May. 2012.

[28] P. Kazanzides, B. Mittelstadt, J. Zuhars, P. Cain, and H. Paul. Surgical and industrial robots: Comparison and case study. In *Proc. of the Intl. Robots and Vision Automation Conf.*, page 10, Detroit, MI, Apr. 1993.

[29] P. Kazanzides, J. Zuhars, B. Mittelstadt, B. Williamson, P. Cain, F. Smith, L. Rose, and B. Musits. Architecture of a surgical robot. In *IEEE Intl. Conf. on Systems, Man and Cybernetics*, volume 2, pages 1624–1629, Oct. 1992.

[30] T. P. Kelly. *Arguing safety - A Systematic Approach to Managing Safety Cases*. PhD thesis, Department of Computer Science, University of York, 1998.

[31] N. Leveson. A new accident model for engineering safer systems. *Safety Science*, 42(4):237–270, 2004.

[32] N. Leveson and T. Shimeall. Safety assertions for process-control systems. In *Intl. Symp. on Fault Tolerant Computing*, pages 236–240, Milan, Jul. 1983. IEEE.

[33] N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[34] R. Muradore, D. Bresolin, L. Geretti, P. Fiorini, and T. Villa. Robotic surgery - formal verification of plans. *IEEE Robotics & Automation Magazine*, 18(3):24–32, Sep. 2011.

[35] C. Pons, R. Giandini, and G. Arévalo. A systematic review of applying modern software engineering techniques to developing robotic systems. *INGENIERÍA E INVESTIGACIÓN*, 32(1):58–63, 2012.

[36] R. Rengasamy. *A framework for integrating process monitoring, diagnosis and supervisory control*. PhD thesis, Purdue University, 1995.

[37] A. Rovetta. Telerobotic surgery control and safety. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 3, pages 2895–2900, 2000.

[38] H. M. Shao, J. Y. Chen, T. K. Truong, I. S. Reed, and Y. S. Kwoh. A New CT-Aided Robotic Stereotaxis System. In *Annu. Symp. Comput. Appl. Med Care.*, volume 13, pages 668–672, Nov. 1985.

[39] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, second edition, 2002.

[40] R. Taylor. A Perspective on Medical Robotics. *Proceedings of the IEEE*, 94(9):1652–1664, Sep. 2006.

[41] R. Taylor, J. Funda, B. Eldridge, S. Gomory, K. Gruben, D. LaRose, M. Talamini, L. Kavoussi, and J. Anderson. A telerobotic assistant for laparoscopic surgery. *IEEE Engineering in Medicine and Biology Magazine*, 14(3):279–288, May/Jun. 1995.

[42] R. Taylor, P. Jensen, L. Whitcomb, A. Barnes, R. Kumar, D. Stoianovici, P. Gupta, Z. Wang, E. Dejuan, and L. Kavoussi. A steady-hand robotic system for microsurgical augmentation. *Intl. J. of Robotics Research*, 18(12):1201–1210, 1999.

[43] R. Taylor, P. Kazanzides, B. Mittelstadt, and H. Paul. Redundant consistency checking in a precise surgical robot. In *Proc. of the 12th Annual Intl. Conf. of the IEEE*, pages 1933–1935, Nov. 1990.

[44] R. Taylor, B. Mittelstadt, H. Paul, W. Hanson, P. Kazanzides, J. Zuhars, B. Williamson, B. Musits, E. Glassman, and W. Bargar. An image-directed robotic system for precise orthopaedic surgery. *IEEE Trans. on Robotics and Automation*, 10(3):261–275, Jun. 1994.

[45] R. Taylor, H. Paul, P. Kazanzides, B. Mittelstadt, W. Hanson, J. Zuhars, B. Williamson, B. Musits, E. Glassman, and W. Bargar. Taming the bull: Safety in a precise surgical robot. In *Intl. Conf. on Advanced Robotics (ICAR)*, volume 1, pages 865–870, Jun. 1991.

[46] R. Taylor and D. Stoianovici. Medical robotics in computer-integrated surgery. *IEEE Trans. on Robotics and Automation*, 19(5):765–781, Oct. 2003.

[47] P. Varley. Techniques for development of safety-related software for surgical robots. *IEEE Trans. on Information Technology in Biomedicine*, 3(4):261–267, Dec. 1999.

[48] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. Kavuri. A review of process fault detection and diagnosis, Part I: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293–311, 2003.

[49] M. L. Visinsky, J. R. Cavallaro, and I. D. Walker. Expert system framework for fault detection and fault tolerance in robotics. *Computers and Electrical Engineering*, 20(5):421–435, 1994.

[50] K. Wika. *Safety kernel enforcement of software safety policies*. PhD thesis, University of Virginia, May 1995.

[51] Y. Zhang and J. Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2):229–252, 2008.